

UNIVERSIDADE FEDERAL DO MARANHÃO
CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA DE ELETRICIDADE

Ariel Soares Teles

*AutonomicSec: Um Mecanismo Autônomo para Segurança de
Redes baseado em Decepção*

São Luís
2012

Ariel Soares Teles

*AutonomicSec: Um Mecanismo Autônomo para Segurança de
Redes baseado em Decepção*

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia de Eletricidade da Universidade Federal do Maranhão como requisito parcial para a obtenção do grau de MESTRE em Engenharia de Eletricidade.

Orientador: Zair Abdelouahab

Doutor em Computer Science – UFMA

São Luís

2012

Teles, Ariel Soares

AutonomicSec: Um Mecanismo Autônomo para Segurança de Redes baseado em Decepção / Ariel Soares Teles. – São Luís, 2012.

114 f.

Orientador: Zair Abdelouahab.

Impresso por computador (fotocópia).

Dissertação (Mestrado) – Universidade Federal do Maranhão, Programa de Pós-Graduação em Engenharia de Eletricidade. São Luís, 2012.

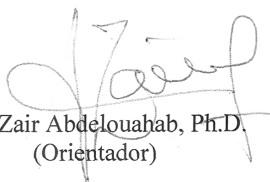
1. Segurança de Redes. 2. Computação Autônoma. 3. Metodologia de Decepção. I. Abdelouahab, Zair, orient. II. Título.

CDU 004.056.53

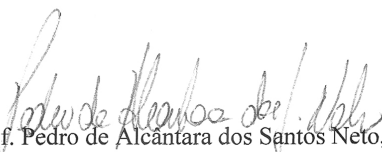
AUTONOMICSEC: UM MECANISMO AUTONÔMICO PARA SEGURANÇA DE REDES BASEADO EM DECEPÇÃO

Ariel Soares Teles

Dissertação aprovada em 21 de março de 2012.



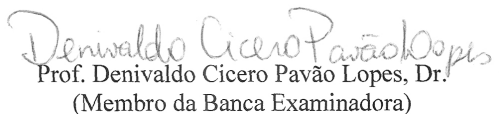
Prof. Zair Abdelouahab, Ph.D.
(Orientador)



Prof. Pedro de Alcântara dos Santos Neto, Dr.
(Membro da Banca Examinadora)



Prof. Francisco José da Silva e Silva, Dr.
(Membro da Banca Examinadora)



Prof. Denivaldo Cicero Pavão Lopes, Dr.
(Membro da Banca Examinadora)

*Dedico esta dissertação de
mestrado à minha mãe, maior
fonte de inspiração.*

Resumo

Segurança em redes de computadores compreende a área responsável pela proteção dos dados que a transitam. A busca por melhores estratégias de segurança tem aumentado consideravelmente, tendo em vista a grande quantidade de tentativas de ataques que vem sendo realizados. Esses ataques tem causado prejuízos financeiros e de imagem para empresas, instituições e pessoas físicas. Vários obstáculos a serem enfrentados para se alcançar redes realmente seguras existem e isso eleva a complexidade do problema da gerência de segurança. Por esse motivo é interessante a utilização de recursos oferecidos pela Computação Autônômica (CA). Sistemas de CA são capazes de gerenciarem a si próprios e se adaptarem dinamicamente às mudanças a fim de restabelecer seu equilíbrio de acordo com as políticas e os objetivos de negócio. A arquitetura e as propriedades de CA para a implementação de sistemas propõe uma abordagem com muitas vantagens para ser aplicada à segurança de redes. Neste trabalho, apresentamos os conceitos de CA e mostramos sua aplicabilidade ao contexto de segurança em redes de computadores. A aplicação dos conceitos de CA à segurança de redes introduz no sistema a capacidade de auto-segurança. Para mostrar a viabilidade em conseguir auto-segurança, desenvolvemos e apresentamos um mecanismo autônomo para segurança de redes. Este mecanismo é representado, inicialmente, por um *framework* autônomo, no qual é organizado seguindo o modelo MAPE-K. Neste modelo gerentes autônomos realizam as atividades de sensoriamento do ambiente de execução, análise de contexto, planejamento e execução de ações de reconfiguração dinâmica. Em seguida, implementamos dois ciclos autônomos. O primeiro tem a funcionalidade de gerar regras de *firewall* baseadas em logs de *honeypots*. O segundo ciclo é responsável por manipular dinamicamente *honeypots* virtuais que são considerados comprometidos. Os resultados mostram que é possível obter integração e cooperação entre os sistemas de segurança; inteligência, através da implantação de estratégias autônomas que dinamizam o processo de proteção; e autonomia, para alcançar autossegurança na rede.

Palavras-chaves: Segurança de Redes, Computação Autônômica, Metodologia de Decepção.

Abstract

Security in computer networks is the area responsible for protecting the data passing through it. The research for better security strategies has increased considerably since exists a vast number of attempted attacks. These attacks have caused financial loss and reputation damage to companies, institutions and individuals. There are several obstacles to achieve security into networks and it led to increase the problem complexity of security management. For this reason, it is interesting using the resources offered by Autonomic Computing (AC). AC systems are capable of manage themselves and to adapt dynamically to changes in order to restore its balance according to policies and business goals. The architecture and properties of AC to implement systems offers many advantages to be applied to network security. In this work, we present the concepts of AC and demonstrate its applicability on the network computer security context. The AC concepts application in network security introduces the auto-security capability to the system. To show the feasibility of achieving auto-security, we developed and present an autonomic mechanism for network protection. This mechanism is represented, initially, by an autonomic framework, which is organized according to MAPE-K model. In this model, autonomic managers perform the sensing activities on the execution environment, context analysis, planning and execution of dynamic reconfiguration actions. Then, we implemented two autonomic cycles. The first cycle aims to generate firewall rules based on honeypots log files. The second cycle is responsible for manipulate, dynamically, virtual honeypots that are classified as compromised. The results show that it is possible to obtain integration and cooperation between security systems; intelligence, through the deployment of autonomic strategies that turn the protection process dynamic; and autonomy, to achieve self-security on the network.

Keywords: Network Security, Autonomic Computing, Methodology of Deception.

Agradecimentos

Agradeço a Deus em primeiro lugar e a minha família, que sempre me apoiou em momentos difíceis nesses dois anos de curso, principalmente minha mãe Fátima, que me incentiva nos estudos desde criança, meu pai Marcondes e minha irmã Layla. Agradeço ao meu orientador, Prof. Zair Abdelouahab, que me conduziu na elaboração deste trabalho com inúmeras discussões sobre o tema de pesquisa e também de como fazer pesquisa científica, que contribuíram muito. Mas agradeço principalmente por ele ter acreditado em mim, desde o início, ao me aceitar no Mestrado como seu aluno.

Agradeço a equipe do Laboratório de Sistemas em Arquiteturas Computacionais (LABSAC) da UFMA, com os quais compartilhamos conhecimentos na área de pesquisa, em Segurança de Redes. Em especial, Lianna, Vladimir e Jean “Pablito”, que contribuíram em discussões sobre o assunto.

Agradeço a equipe do Laboratório de Sistemas Distribuídos (LSD) da UFMA, com os quais compartilhamos conhecimentos na área de pesquisa, em Computação Autônoma. Em especial, agradeço a Jesseildo e Berto, que contribuíram efetivamente com este trabalho.

Agradeço ao Prof. Francisco José da Silva e Silva, pelos ensinamentos passados durante suas disciplinas, contribuições na área de Computação Autônoma e por acreditar no meu potencial, ao me aceitar como seu aluno de Doutorado.

Agradeço a minha namorada Milena, pelo companheirismo e também pela paciência e presença em momentos de dificuldade.

Agradeço ao meu parceiro Linhares Júnior, pelas conversas e companheirismo.

Agradeço a Profa. Maria da Guia, pelas conversas sobre pesquisa científica e dicas passadas.

Agradeço a UFMA e ao Programa de Pós-Graduação em Engenharia de Eletricidade, especialmente ao Alcides que sempre se faz presente, atendendo as necessidades dos alunos.

Agradeço a Fundação de Amparo à Pesquisa e ao Desenvolvimento Científico e Tecnológico do Maranhão (FAPEMA), pelo apoio financeiro concedido através de bolsa de estudos.

Por fim, agradeço a meus amigos e colegas, de Parnaíba e os que fiz durante esses dois anos de luta, que contribuíram, de forma direta ou indireta, para a concretização deste trabalho.

“A diferença entre ‘envolvimento’ e ‘comprometimento’ é como ovo com bacon no café da manhã: a galinha estava ‘envolvida’ - o porco, ‘comprometido’.”

Autor Desconhecido

Lista de Figuras

2.1	Ciclo de gerenciamento de sistemas [23].	24
2.2	MAPE-K: Ciclo de gerenciamento autônômico [45].	26
2.3	Definição de uma regra do tipo ECA.	30
3.1	Quatro fases de proteção [97].	36
3.2	Uma máquina virtual [82] [53].	49
3.3	Forma de implementar virtualização [82] [53].	50
4.1	<i>Framework</i> conceitual do ISDS [52].	53
4.2	Neurônio Virtual [27].	54
4.3	Estrutura de organização hierárquica e P2P dos neurônios virtuais [27].	55
4.4	Arquitetura do AND [22].	56
4.5	Distância de Anormalidade [77].	57
5.1	Exemplo de reconfiguração.	64
5.2	Arquitetura do <i>Framework</i>	67
5.3	Comunicação entre componentes do <i>Framework</i>	69
5.4	Arquitetura do Primeiro Ciclo Autônômico.	71
5.5	Topologia do Primeiro Ciclo Autônômico.	72
5.6	Lista Cinza.	73
5.7	Lista Negra.	73
5.8	Arquitetura do Segundo Ciclo Autônômico.	75
5.9	Topologia do Segundo Ciclo Autônômico.	77
6.1	Diagrama de componentes do <i>Framework</i>	80

6.2	Diagrama de classe do <i>Framework</i>	81
6.3	Lógica do sensor no primeiro ciclo autônomo.	85
6.4	Lógica da estratégia autônoma do primeiro ciclo.	86
6.5	Regras para controle de tráfego e log no <i>iptables</i>	88
6.6	Lógica da estratégia autônoma do segundo ciclo.	89
6.7	Cenário para coleta de resultados.	91
6.8	Log gerado no <i>Honeypot</i>	92
6.9	Informações resultantes do filtro realizado pelo monitor.	93
6.10	Arquivo de configuração da lista branca.	93
6.11	Arquivo de configuração dos serviços.	94
6.12	Servidores de produção do cenário.	94
6.13	Regras geradas pela estratégia baseada em intenções.	95
6.14	Regras otimizadas.	96
6.15	Log gerado no <i>Firewall</i>	97
6.16	Endereços após filtro realizado pelo monitor.	97
6.17	Arquivo de configuração dos <i>Honeypots</i>	98
6.18	Plano de ações criado pelo analisador/planejador.	98

Lista de Tabelas

3.1	Comparação entre os tipos de <i>honeypots</i>	40
4.1	Comparação Qualitativa entre Sistemas de Segurança.	61

Lista de Siglas

ACID	Atomicidade, Consistência, Isolamento e Durabilidade.
ACLs	<i>Access Control Lists.</i>
AND	<i>Autonomic Network Defense System.</i>
CIDS	<i>Collaborative Intrusion Detection System.</i>
CMI	<i>Component Management Interface.</i>
CRM	<i>Component Runtime Manager.</i>
DDoS	<i>Distributed Denial of Service.</i>
DNS	<i>Domain Name System.</i>
DoS	<i>Denial of Service.</i>
ECA	Evento-Condição-Ação.
HIDS	<i>Host Based Intrusion Detection System.</i>
ICMP	<i>Internet Control Message Protocol.</i>
IDE	<i>Integrated Development Environment.</i>
IDS	<i>Intrusion Detection Systems.</i>
IGMP	<i>Internet Group Management Protocol.</i>
IP	<i>Internet Protocol.</i>
IPS	<i>Intrusion Prevention System.</i>
IPv4	<i>Internet Protocol version 4.</i>
IPv6	<i>Internet Protocol version 6.</i>
ISDS	<i>Intelligent Security Defensive Software.</i>
MAPE-K	<i>Monitoring, Analysis, Planning, Execution and Knowledge loop.</i>

mDNS	<i>Multicast Domain Name System.</i>
NIDS	<i>Network Based Intrusion Detection System.</i>
P2P	<i>Peer-to-peer.</i>
QoP	<i>Quality of Protection.</i>
QoS	<i>Quality of Service.</i>
SGBD	<i>Sistemas de Gerenciamento de Banco de Dados.</i>
SMS	<i>Short Message Service.</i>
SSH	<i>Secure Shell.</i>
TCP	<i>Transmission Control Protocol.</i>
UDP	<i>User Datagram Protocol.</i>

Sumário

Lista de Figuras	ix
Lista de Tabelas	xi
Lista de Siglas	xii
1 Introdução	17
1.1 Objetivos	19
1.2 Organização do Trabalho	20
2 Computação Autônoma	22
2.1 Considerações Iniciais	22
2.2 Propriedades de Sistemas Autônomos	23
2.3 Arquitetura de um Sistema Autônomo	24
2.4 MAPE-K	25
2.5 Monitoramento	27
2.5.1 Classificação de Sistemas de Monitoramento	27
2.6 Análise e Planejamento	29
2.6.1 Análise e Planejamento Baseados em Regras ECA	29
2.7 Execução de Ações de Reconfiguração	30
2.7.1 Questões de Projeto de Mecanismos de Reconfiguração	31
2.8 Conclusão	32
3 Estado Atual da Segurança em Redes de Computadores	33
3.1 Considerações Iniciais	33

3.2	Fases para Proteção de Redes	35
3.2.1	Prevenção	35
3.2.2	Detecção	38
3.2.3	Defesa	42
3.2.4	Forense	43
3.3	Problemas Enfrentados	43
3.3.1	Problemas em Sistemas de Segurança	44
3.3.2	Robustez dos Ataques a Sistemas Computacionais	46
3.4	Aplicações de Virtualização em Segurança	48
3.5	Conclusão	51
4	Trabalhos Relacionados	52
4.1	Sistemas de Segurança baseados em Computação Autônoma	52
4.2	Propriedades de Computação Autônoma em Sistemas de Segurança	58
4.3	Caracterização da Proposta	59
4.4	Conclusão	62
5	Segurança Autônoma em Redes de Computadores	63
5.1	Aplicabilidade	63
5.2	Uma proposta de mecanismo autônomo para Segurança de Redes	66
5.2.1	<i>Framework</i> Autônomo	67
5.2.2	Primeiro Ciclo Autônomo	70
5.2.3	Segundo Ciclo Autônomo	74
5.2.4	Discussão	77
5.3	Conclusão	79
6	Avaliação do Mecanismo Autônomo	80
6.1	Implementação do <i>Framework</i> Autônomo	80

6.2	Avaliação dos Ciclos Autônômicos	84
6.2.1	Testes	85
6.2.2	Resultados	90
6.3	Conclusão	98
7	Conclusões e Trabalhos Futuros	100
	Referências Bibliográficas	105

1 Introdução

Segurança em redes de computadores compreende a área responsável pela proteção dos dados que a transitam contra alterações indevidas, acesso não autorizado e indisponibilidade. Desde o surgimento da Internet, a busca por melhores estratégias de segurança tem aumentado consideravelmente, tendo em vista a grande quantidade de tentativas de ataques que vem sendo realizados. Esses ataques, quando bem sucedidos, tem causado prejuízos financeiros e de imagem para empresas, instituições e pessoas físicas.

Vários obstáculos a serem enfrentados para se alcançar redes realmente seguras existem, dentre eles pode-se destacar a existência de dependência dos sistemas de segurança por gerenciamento com intervenção humana, sendo este um processo que aumenta continuamente o nível de dificuldade. O fato dos ataques à sistemas computacionais estarem se tornando cada vez mais sofisticados e as várias deficiências encontradas nos atuais sistemas de segurança também são outros exemplos de obstáculos [97].

Tudo isso eleva a complexidade do problema da gerência de segurança e por esse motivo é interessante a utilização de recursos oferecidos pela Computação Autônômica (CA) [81]. Sistemas de CA são capazes de gerenciarem a si próprios e se adaptarem dinamicamente às mudanças a fim de restabelecer seu equilíbrio de acordo com as políticas e os objetivos de negócio. Para isso, dispõem de mecanismos efetivos que os permitem monitorar, controlar e regular a si próprios, bem como recuperarem-se de problemas sem a necessidade de intervenções externas.

A arquitetura e as propriedades de CA para a implementação de sistemas propõe uma abordagem com muitas vantagens para ser aplicada à segurança de redes [52]. Além de apresentar características intrínsecas de auto-gerenciamento, um elemento autônômico provê outras funcionalidades que podem ser utilizadas para resolver problemas particulares à segurança de redes, com o uso de técnicas de aprendizagem e cooperação entre as aplicações.

Neste trabalho, apresentamos os conceitos de CA e mostramos sua aplicabilidade ao contexto de segurança em redes de computadores. A aplicação dos conceitos de CA à segurança de redes introduz no sistema a capacidade de auto-segurança, através da qual serviços e funções de gerenciamento da segurança são executados sem a necessidade de um gerente humano, a partir apenas da definição de objetivos e parâmetros iniciais fornecidos por seus administradores.

Para mostrar a viabilidade em conseguir auto-segurança, desenvolvemos e apresentamos um mecanismo autônomo para proteção de redes. Este mecanismo é representado, inicialmente, por um *framework* autônomo, no qual é organizado seguindo o modelo *Monitoring, Analysis, Planning, Execution and Knowledge loop* (MAPE-K). Neste modelo gerentes autônicos realizam as atividades de sensoriamento do ambiente de execução, análise de contexto, planejamento e execução de ações de reconfiguração dinâmica.

Em seguida, para mostrar a aplicabilidade do *framework* e também gerando contribuições para resolver problemas específicos de segurança de redes, implementamos dois ciclos autônicos. O primeiro tem a funcionalidade de gerar regras de *firewall* baseadas em logs de *honeypots*. Ou seja, os dados sensoreados são os logs a nível de rede gerados por *honeypots* de baixa interatividade e a os atuadores aplicam regras para controle de tráfego entre redes distintas em uma aplicação de *firewall*. As regras são geradas segundo uma estratégia autônoma baseada em intenções, que está associada a modificação das permissões de acesso de usuários externos a serviços da rede interna de acordo com sua interação com *honeypots*.

O segundo ciclo é responsável por manipular dinamicamente *honeypots* virtuais que são considerados comprometidos. Um *honeypot* é considerado comprometido quando o intruso que interage com ele descobre que está sendo enganado, ganha domínio sob este e passa a utilizá-lo como intermediador para a realização de outros ataques. O componente sensor neste ciclo irá fazer a análise em logs de *firewall* para verificar a ocorrência de tráfego de saída originados a partir de *honeypots* e, caso ocorra, a máquina virtual comprometida será substituída por sua réplica. Esta última é criada inicialmente antes do *honeypot* ser colocado em atividade, não havendo a possibilidade de estar comprometida. A ação executada neste ciclo é a própria substituição dos *honeypots* comprometidos.

Esses dois ciclos autônômicos desenvolvidos tem o intuito de propor soluções para problemas particulares em segurança de redes, mais especificamente na área de metodologias de decepção [76]. Estas últimas são definidas em [76] como a criação de um ambiente falso para enganar usuários mal intencionados. Os ciclos são extensões do *framework* desenvolvido. No entanto, consideramos que, de uma forma geral, o escopo deste trabalho envolve a aplicabilidade dos conceitos que permeiam a CA para a área de segurança de redes.

Os resultados mostram que, com a utilização de CA e extensão do *framework*, é possível obter integração e cooperação entre os sistemas de segurança que foram inicialmente desenvolvidos para trabalharem isoladamente, inteligência através da implantação de estratégias autônômicas que dinamizam o processo de proteção e, por fim, autonomia para alcançar auto-segurança na rede, de maneira que os sistemas consigam se auto-gerenciarem.

1.1 Objetivos

Esta dissertação de mestrado tem como objetivo geral propor um mecanismo autônômico para segurança de redes baseado em decepção. A intenção de nossa proposta é aumentar o nível de segurança da rede, fazendo que os sistemas tenham autonomia para realizar a defesa da rede contra atividades maliciosas e, através disso, seja possível introduzir a ela a capacidade de auto-segurança. Para isso, considera-se os seguintes objetivos específicos:

- Mostrar de forma argumentativa, e também através de exemplos, as maneiras que conceitos de Computação Autônômica podem ser aplicados a segurança de redes;
- Fornecer autonomia para sistemas que atuam na área de segurança de redes;
- Elaborar um mecanismo autônômico para a proteção de redes, através do projeto e implementação de um *framework* que pode ser estendido para solucionar problemas de integração, cooperação e inteligência;

- Desenvolver um ciclo autônomo responsável por dinamizar o processo de análise dos logs de *honeypots* e utilizar as informações coletadas para a criação de filtros de tráfego, com a aplicação de regras de *firewall*;
- Propor uma solução para o problema de *honeypots* comprometidos com a utilização de recursos virtualizados, através de um ciclo autônomo.

1.2 Organização do Trabalho

O restante deste trabalho está estruturado da seguinte forma:

- O **Capítulo 2** descreve todos os fundamentos de CA, mostrando suas propriedades, arquitetura e detalhes do ciclo autônomo, explicando a função de cada fase com informações pertinentes para o contexto de segurança de redes;
- No **Capítulo 3** são descritos os conceitos básicos da área de segurança em redes de computadores e mostrado o estado atual em que elas se encontram nesse segmento. As possíveis fases que uma rede pode ser protegida e as ferramentas e técnicas mais utilizadas em cada uma dessas fases são explicadas, como também os problemas enfrentados por elas no quesito segurança. Ainda neste capítulo, os fundamentos conceituais sobre máquinas virtuais são apresentados, mostrando as principais possibilidades de uso de virtualização na área de segurança.
- O **Capítulo 4** apresenta os trabalhos relacionados relevantes e uma caracterização de nossa proposta, os quais são organizados de forma a mostrar claramente as contribuições dessa dissertação;
- O **Capítulo 5** mostra a necessidade que a segurança em redes de computadores tem por mecanismos autônomos, argumentando sua aplicabilidade. Uma proposta de mecanismo autônomo para segurança de redes é feita, composta de um *framework* e dois ciclos autônomos utilizados para fornecer autonomia a esta área. As funcionalidades e modelagem da proposta são apresentadas, como também o contexto em que ela se aplica;

-
- No **Capítulo 6** é dada continuidade com a explicação da contribuição, descrevendo a sua implementação, tecnologias utilizadas e resultados alcançados;
 - O **Capítulo 7** apresenta as conclusões, contribuições, lições aprendidas, limitações e possibilidades de trabalhos a serem desenvolvidos futuramente.

2 Computação Autônômica

Este capítulo apresenta uma introdução a Computação Autônômica, mostrando sua origem e seus conceitos, suas principais propriedades, arquitetura, o modelo MAPE-K e todas suas fases, que correspondem ao ciclo de gerenciamento autônômico. Neste também é detalhado cada fase do ciclo com informações pertinentes ao contexto da segurança de redes.

2.1 Considerações Iniciais

O termo Computação Autônômica surgiu em 2001, com um manifesto publicado por Paul Horn, pesquisador da IBM, que lançou um desafio sobre o problema da crescente complexidade de gerenciamento do software [46]. O termo autônômico vem da biologia e está relacionado às reações fisiológicas involuntárias do sistema nervoso [63]. No corpo humano, o sistema nervoso autônômico cuida de reflexos inconscientes, isto é, de funções corporais que não requerem nossa atenção como a contração e expansão da pupila, funções digestivas do estômago e intestino, a frequência e profundidade da respiração, a dilatação e constrição de vasos sanguíneos, etc. Esse sistema reage às mudanças, ou perturbações, causadas pelo ambiente através de uma série de modificações, a fim de conter as perturbações causadas ao seu equilíbrio interno.

Em analogia ao comportamento humano, diz-se que um sistema computacional está em equilíbrio quando o seu ambiente interno (formado pelos seus subsistemas e pelo próprio sistema) está em proporção devida com o ambiente externo. Conforme observaram Parashar e Hariri em [68] e detalhado em [69], se o ambiente interno ou externo perturba a estabilidade do sistema, ele sempre atuará de modo a recuperar o equilíbrio original. Dessa forma, os sistemas de CA são sistemas capazes de se auto-gerenciarem e se adaptarem dinamicamente às mudanças a fim restabelecer seu equilíbrio de acordo com as políticas e os objetivos de negócio do sistema [46]. Para isso, devem dispor de mecanismos efetivos que os permita monitorar, controlar

e regular a si próprios, bem como recuperarem-se de problemas sem a necessidade de intervenções externas.

2.2 Propriedades de Sistemas Autônomicos

A essência da CA é o auto-gerenciamento. Para implementá-lo, o sistema deve ao mesmo tempo estar atento a si próprio e ao seu ambiente. Desta forma, o sistema deve conhecer com precisão a sua própria situação e ter consciência do ambiente operacional em que atua. Do ponto de vista prático, conforme Hariri [40], o termo computação autônoma tem sido utilizado para denotar sistemas que possuem as seguintes propriedades:

- **Autoconsciência (*self-awareness*):** O sistema conhece a si próprio: seus componentes e inter-relações, seu estado e comportamento;
- **Consciência do contexto (*context-aware*):** O sistema deve ser ciente do contexto de seu ambiente de execução e ser capaz de reagir a mudanças em seu ambiente;
- **Autoconfiguração (*self-configuring*):** O sistema deve ajustar dinamicamente seus recursos baseado em seu estado e no estado do ambiente de execução;
- **Auto-otimização (*self-optimizing*):** O sistema é capaz de detectar degradações de desempenho e de realizar funções para auto-otimização;
- **Autoproteção (*self-protecting*):** O sistema é capaz de detectar e proteger seus recursos de atacantes internos e externos, mantendo sua segurança e integridade geral;
- **Autocura (*self-healing*):** O sistema deve possuir a habilidade de identificar potenciais problemas e de se reconfigurar de forma a continuar operando normalmente;
- **Aberto (*open*):** O sistema deve ser portátil para diversas arquiteturas de hardware e software e, conseqüentemente, deve ser construído a partir de protocolos e interfaces abertos e padronizados;

- **Capacidade de Antecipação (*anticipatory*):** O sistema deve ser capaz de antecipar, na medida do possível, suas necessidades e comportamentos considerando seu contexto e de se autogerenciar de forma pró-ativa.

As propriedades de autoconfiguração, auto-otimização, autocura e autoproteção são suficientes para realizar a visão original da Computação Autônoma [63]. Para a incorporação das propriedades de auto-otimização e autocura, mecanismos de autoconsciência, consciência de contexto e autoconfiguração devem ser requisitos do sistema.

Existem ainda outras propriedades que fornecem autonomia para sistemas, como as listadas em [74]: autoaprendizagem, auto-regulação, auto-interesse, auto-criação, auto-replicação, auto-evolução, autodescoberta, entre outras. Para Poslad [74], as propriedades utilizadas para provê autonomia ao sistema dependem do seu domínio e também do seu próprio projeto.

2.3 Arquitetura de um Sistema Autônomo

Arquiteturas de sistemas autônomos visam formalizar um quadro de referência que identifica as funções comuns e estabelece os alicerces necessários para alcançar a autonomia. Em geral, essas arquiteturas apresentam soluções para automatizar o ciclo de gerenciamento de sistemas, conforme mostrado na Figura 2.1, o qual envolve as seguintes atividades:

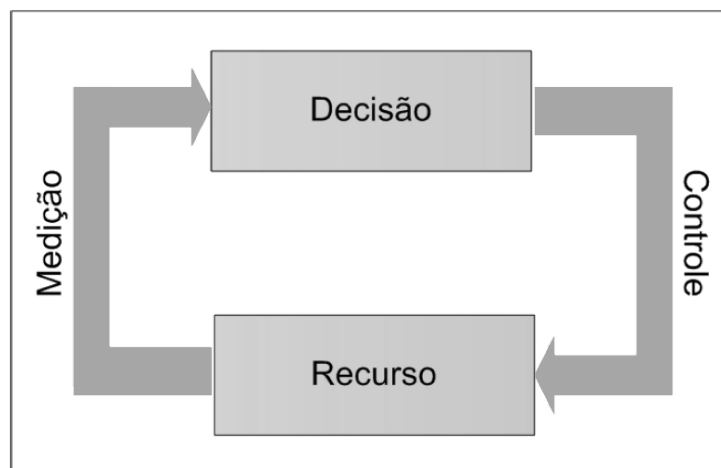


Figura 2.1: Ciclo de gerenciamento de sistemas [23].

- **Monitoramento ou Medição:** Coleta, agrega, correlaciona e filtra dados sobre recursos gerenciados;
- **Análise e Planejamento:** Analisa os dados coletados e determina se devem ser realizadas mudanças nas estratégias utilizadas pelo recurso gerenciado;
- **Controle e Execução:** Escalona e executa as mudanças identificadas como necessárias pela função de análise e decisão.

2.4 MAPE-K

Em 2003 a IBM propôs uma versão automatizada do ciclo de gerenciamento de sistemas chamado de MAPE-K [46], representado na Figura 2.2. Este modelo está sendo cada vez mais utilizado para inter-relacionar os componentes arquiteturais dos sistemas autônômicos. De acordo com essa arquitetura, um sistema autônômico é formado por um conjunto de elementos autônômicos.

Um elemento autônômico contém um único gerente autônômico que representa e monitora um ou mais elementos gerenciados (componente de hardware ou de software) [48]. Cada elemento autônômico atua como um gerente responsável por promover a produtividade dos recursos e a qualidade dos serviços providos pelo componente do sistema no qual está instalado.

No ciclo MAPE-K autônômico (Figura 2.2), o elemento gerenciado representa qualquer recurso de software ou hardware o qual é dado o comportamento autônômico através do acoplamento de um gerenciador autônômico, podendo ser por exemplo um servidor Web ou banco de dados, um componente de software específico em um aplicativo (por exemplo, o otimizador de consulta em um banco de dados), o sistema operacional, um conjunto de máquinas em um ambiente de rede, etc.

Os sensores (*sensors*) são responsáveis por coletar informações do elemento gerenciado. Estes dados podem ser os mais diversos possíveis, por exemplo, o tempo de resposta das requisições dos clientes, caso o elemento gerenciado seja um servidor Web. As informações coletadas pelos sensores são enviadas aos monitores (*monitors*) onde são interpretadas, preprocessadas e colocadas em um nível mais alto de abstração, para então serem enviadas para a etapa seguinte no ciclo, a fase de análise

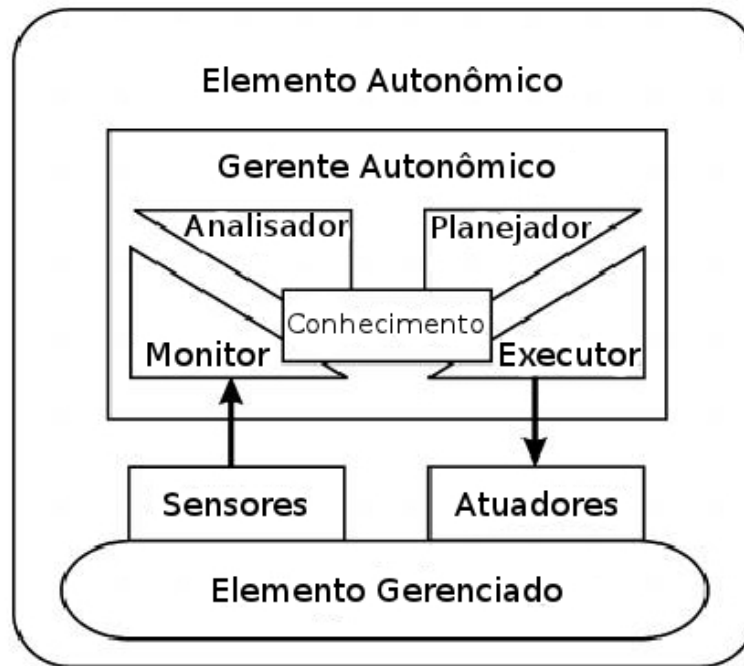


Figura 2.2: MAPE-K: Ciclo de gerenciamento autônomo [45].

e planejamento (*analyze and planing*). Nesta fase, tem-se como produto uma espécie de plano de trabalho, que consiste de um conjunto de ações a serem executadas pelo executor (*execute*).

O componente responsável por fazer as alterações no ambiente é chamado de atuador (*effectors*). Somente os sensores e atuadores possuem acesso direto ao elemento gerenciado. Durante todo ciclo de gerenciamento autônomo, pode haver a necessidade de uma tomada de decisão, dessa forma, faz-se necessário também a presença de uma base de conhecimento (*knowledge*), sendo que esta é comumente mais explorada na fase de análise e planejamento [35].

Este modelo é implementado utilizando dois ou mais ciclos de gerenciamento autônomo, um ou mais ciclos de controle local e um global. Os ciclos de controle local tratam apenas de estados conhecidos do ambiente local, sendo baseado no conhecimento encontrado no próprio elemento gerenciado. Por esta razão, o ciclo local é incapaz de controlar o comportamento global do sistema. O ciclo global, por sua vez, a partir de dados provenientes dos gerenciadores locais ou através de um monitoramento a nível global, podem tomar decisões e atuar globalmente no sistema. No entanto, a implementação das interações entres os diversos níveis existentes vai depender das necessidades e das limitações da aplicação.

2.5 Monitoramento

O monitoramento corresponde a primeira fase do ciclo autônomo MAPE-K. Nesta etapa, sensores são utilizados com a finalidade de obter dados que reflitam mudanças de comportamento do elemento gerenciado ou informações do ambiente de execução que sejam relevantes ao processo de auto-gerenciamento. Os sensores são dispositivos de hardware ou software que estão diretamente ligados ao componente que se deseja monitorar.

O tipo de informação coletada bem como o modelo do(s) monitor(es) empregados são específicos para cada tipo de aplicação. Contudo, alguns requisitos são comuns entre eles, tais como a filtragem, a escalabilidade, a dinamicidade e a tolerância a falhas [58]. Um monitor deve prover filtragem, pois a quantidade de dados coletados pode crescer muito rapidamente e consumir recursos de transmissão, processamento e armazenamento. Grande parte desses dados podem ser de pouca relevância e, portanto, descartá-los não implicaria em prejuízo. Considerando-se que um sistema pode crescer indefinidamente, contendo inúmeros objetos sob monitoramento, a tarefa dos monitores não pode consumir recursos e reduzir o desempenho do sistema. Em contrapartida, as mudanças no ambiente ou no comportamento do sistema não podem afetar o serviço de monitoramento [36]. Além disso, falhas podem ocorrer e o monitor pode apresentar algum comportamento atípico ou mesmo parar de funcionar completamente. Dessa forma, devem ser levadas em consideração provisões como redundância e mecanismos de recuperação de falhas dos monitores.

2.5.1 Classificação de Sistemas de Monitoramento

Aplicações autônomas possuem requisitos de monitoramento diferentes, variando os elementos que se deseja monitorar. De acordo com as características deste elementos podem ser implementadas formas de monitoramento específicas. Em [45], Huebscher e McCann identificaram dois tipos de monitoramento em sistemas autônomos, classificados de acordo com o tipo de sensor utilizado:

- **Monitoramento Passivo:** No monitoramento passivo os dados obtidos de monitoramento são fornecidos pelo sistema sem necessidade de nenhuma

alteração na aplicação. Por exemplo, no Linux, informações sobre uso de CPU e memória podem ser coletados do diretório `/proc`;

- **Monitoramento Ativo:** Para que se possa fazer a captura de dados nesse tipo de monitoramento é necessário alterar a implementação da aplicação ou sistema operacional, sendo então considerado um monitoramento invasivo. A exemplo desse tipo de monitoramento são os sensores inseridos na forma de *bytecodes* em aplicações Java.

Conforme observado nos trabalhos de [58] e [45], foi possível constatar que o monitor também pode ser classificado quanto à estratégia utilizada:

- **Monitoramento orientado à eventos:** Eventos são ações que acontecem no sistema e podem alterar o seu estado, por exemplo o “processo ocioso”, “processo em execução”, “início do processo”, “chegada de uma requisição”, etc. Os eventos acontecem instantaneamente e portanto nesse tipo de monitoramento cada evento corresponde a um dado que é transmitido para o monitor. Essa abordagem pode ser aplicada nos casos em que a taxa de ocorrência de eventos é muito baixa. A desvantagem é que o número de eventos gerados pode ser muito grande e informações redundantes e desnecessárias estariam sobrecarregando os recursos do sistema;
- **Monitoramento orientado ao tempo:** Nessa estratégia, o monitor coleta os dados de forma periódica, ou seja, um intervalo de tempo é definido para que o monitor consulte as informações do ambiente em busca de eventos significativos. A grande dificuldade encontrada nessa estratégia está em definir o melhor intervalo para a realização da coleta. A desvantagem nesse caso seria que informações importantes seriam perdidas durante esse intervalo;
- **Monitoramento autônomo:** As duas estratégias acima podem ser intercaladas conforme ocorrem as mudanças no ambiente. Por exemplo, se a taxa de ocorrência de eventos está elevada a melhor estratégia seria a orientada ao tempo. O tamanho do intervalo de tempo na segunda estratégia também pode ser refinado segundo uma abordagem autônoma de acordo com as características do ambiente.

2.6 Análise e Planejamento

A análise vem após o monitoramento no ciclo de gerenciamento MAPE-K, tendo como fase seguinte o planejamento. Em geral estas duas fases são geralmente implementadas em um único componente. O processo de análise e planejamento é essencial para autonomia do sistema, pois é nele que são geradas as decisões sobre quais modificações serão realizadas no sistema, que corresponde ao elemento gerenciado.

A fase de análise e planejamento recebe como entrada os eventos e seus dados gerados pelo sistema de monitoramento e gera como saída um conjunto de ações, também chamado de plano de ações. Estas ações correspondem às operações de reconfiguração que, de acordo com o mecanismo de tomada de decisão, devem ser executadas no sistema a fim de manter o equilíbrio do sistema considerando seus objetivos. Muitas técnicas podem ser empregadas na fase de análise e planejamento, entre as quais se destacam o uso de funções de utilidade, aprendizado por reforço e regras Evento-Condição-Ação (ECA), sendo esta última detalhada a seguir.

2.6.1 Análise e Planejamento Baseados em Regras ECA

Regras do tipo ECA determinam as ações a serem realizadas quando um evento ocorre desde que certas condições sejam satisfeitas. *ECA rules* são especificações declarativas de regras ou regulamentações que determinam o comportamento de componentes de aplicações [23]. *ECA rules* consistem de eventos, condições e ações. Em analogia, outros termos podem ser utilizados: a *trigger* [13] em Sistemas de Gerenciamento de Banco de Dados (SGBD) e o *signal handler* [89] em Sistemas Operacionais.

Em SGBDs, uma vez que o conjunto de regras é definido, o sistema monitora os eventos relevantes. Sempre que se detecta a ocorrência de um evento, o componente responsável pela execução da regra é notificado. Essa notificação é chamada de sinalização do evento. Posteriormente, todas as regras que são definidas para responder a esse evento são executadas. A execução de uma regra contém a avaliação de um estado e execução de uma ação. Primeiro a condição é avaliada e, caso for satisfeita, a ação é executada.

De maneira similar, em Sistemas Operacionais, um sinal é uma forma de comunicação entre processos, sendo uma notificação enviada para um processo com o objetivo de avisá-lo de ocorrências de eventos. Com isso, um processo pode registrar uma operação de tratamento de eventos oriundos de um sinal, o chamado *Signal Handler*, se não for desejado que a operação padrão para este sinal seja executada.

Para cada evento é definido um conjunto de regras que podem gerar uma ou mais ações. Esses eventos também podem satisfazer as condições de diferentes regras e algumas dessas regras podem gerar ações que conflitam entre si. Nem sempre esses conflitos podem ser detectados durante a escrita das regras, sendo muitas vezes descobertos em tempo de execução.

Em *ECA rules* o evento especifica quando as regras deverão ser acionadas. A condição é a parte a ser verificada, podendo esta ser satisfeita ou não. E por fim, a ação, a qual representa a operação a ser realizada caso a condição seja satisfeita. *ECA rules* são definidas da seguinte forma:

on event if condition do action

Figura 2.3: Definição de uma regra do tipo ECA.

2.7 Execução de Ações de Reconfiguração

Na etapa de execução do ciclo MAPE-K são realizadas reconfigurações no sistema de forma a restabelecer seu equilíbrio. A finalidade da reconfiguração é permitir que um sistema evolua (ou simplesmente mude) incrementalmente de uma configuração para outra em tempo de execução, introduzindo pouco (ou, idealmente, nenhum) impacto sobre a execução do sistema. Desta forma, os sistemas (ou aplicações) não necessitam ser finalizados, ou reiniciados, para que haja as mudanças.

A autoconfiguração (*self-configuring*) é a característica do sistema autônomo que o permite ajustar-se automaticamente às novas circunstâncias percebidas em virtude do seu próprio funcionamento, de forma a atender a objetivos especificados pelos processos de autocura, auto-otimização ou autoproteção [23].

O processo de reconfiguração é realizado pelos executores através dos atuadores. Executores recebem como entrada um plano de ações gerado na etapa

de análise e planejamento e utiliza os atuadores pertinentes para implementar as ações de reconfiguração descritas no plano. As reconfigurações devem ser realizadas dinamicamente, sem impor a necessidade de parar e/ou reiniciar o sistema.

Segundo [60], duas abordagens gerais têm sido utilizadas para atingir adaptação: adaptação paramétrica e adaptação composicional. A adaptação paramétrica consiste na alteração de variáveis que determinam o comportamento do sistema. Já a adaptação composicional (ou estrutural) consiste na substituição de algoritmos ou partes estruturais de um software, permitindo que este adote novas estratégias (algoritmos) para tratar situações que não foram inicialmente previstas na sua construção.

2.7.1 Questões de Projeto de Mecanismos de Reconfiguração

O desenvolvimento de um mecanismo de reconfiguração dinâmica não é algo simples. Diversos requisitos devem ser considerados a fim de manter as características e funcionalidades do sistema [66], entre as quais destaca-se:

- **Separação de responsabilidades:** No desenvolvimento de sistemas autônomicos, a separação de responsabilidades permite que o código funcional da aplicação (responsável pelas regras de negócio) seja separado do código responsável pela adaptação. Isto simplifica o desenvolvimento, a manutenção e o reuso do código adaptativo;
- **Confiabilidade:** Um problema quando se modifica um sistema em tempo de execução é a sincronização entre reconfigurações e execução funcional do sistema. A reconfiguração não deve prejudicar a funcionalidade do sistema. Léger et al., em [55], definiram um conjunto de propriedades a fim de garantir a confiabilidade no contexto das reconfigurações dinâmicas em sistemas baseados em componentes. Esse conjunto de propriedades foi baseado em transações e denominado Atomicidade, Consistência, Isolamento e Durabilidade (ACID);
- **Preservação de consistência:** Quando a reconfiguração do sistema ocorre em tempo de execução, é importante que a reconfiguração preserve a consistência do sistema. O estado do sistema interno deve ser mantido e as informações trocadas entre os componentes não devem ser perdidas;

- **Custo da reconfiguração:** Em [39] o custo de reconfiguração é definido como uma medida dos efeitos negativos introduzidos pela reconfiguração. Esses efeitos negativos incluem, por exemplo, a indisponibilidade temporária do serviço, ou a perturbação induzida em outros serviços após o possível aumento do consumo de recursos de rede durante a reconfiguração. A relação custo/benefício deve ser avaliada.

2.8 Conclusão

Nesse capítulo foi visto que o auto-gerenciamento é a característica necessária para um sistema ser classificado como autônomo. Foi mostrado as principais propriedades de sistemas autônomos e que para um sistema alcançar o auto-gerenciamento é necessário provê as propriedades de auto-configuração, auto-otimização, autoproteção, autocura, autoconsciência e ciência de contexto.

Também apresentou-se a arquitetura de um sistema autônomo e, em seguida, o modelo MAPE-K, proposto originalmente pela IBM e que é a referência mais utilizada para o desenvolvimento de sistemas autônomos. Por fim, foi descrito as fases que envolvem o ciclo do gerenciamento autônomo, mostrando detalhes particulares de cada uma.

3 Estado Atual da Segurança em Redes de Computadores

Este capítulo mostra a situação atual em que se encontram as redes de computadores com relação a sua segurança. Em uma visão de maior nível, são mostradas as possíveis fases que uma rede pode ser protegida e as ferramentas e técnicas mais utilizadas em cada uma dessas fases. Em seguida, os atuais problemas de segurança que as redes enfrentam são descritos. Por fim, os fundamentos conceituais sobre máquinas virtuais são apresentados, mostrando as principais possibilidades de uso de virtualização na área de segurança.

3.1 Considerações Iniciais

O grande crescimento na quantidade de componentes e serviços oferecidos nas atuais redes de computadores tem aumentado o nível de complexidade no trabalho de gerenciamento e administração destas. Cada vez mais são integrados novos dispositivos às redes, que requerem conectividade a qualquer momento e em qualquer lugar, além de se caracterizarem por sua heterogeneidade que dificulta o processo de padronização das aplicações.

Então se faz necessário aplicar a ideia de CA às Redes de Computadores, através da atribuição da capacidade para gerenciarem a si próprias, denominadas então como Redes Autônomicas [19]. Os serviços e funções de gerenciamento da rede são executados sem a necessidade de um gerente humano de maneira transparente para seus usuários, havendo apenas a necessidade de objetivos e parâmetros iniciais, os quais o sistema leva em consideração. A rede deve ser capaz de aprender com as ações dos seus componentes através da análise dos resultados obtidos. A capacidade de adaptação e aprendizagem são as características das redes autônomicas.

Neste cenário, não se pode deixar de lado a segurança da informação, que é requisito fundamental para o bom funcionamento de qualquer sistema computacional,

compreendendo o conjunto de medidas que visam preservar e proteger as informações e os sistemas de informação. Visto que essas redes estão conectadas quase sempre à Internet, as aplicações residentes nelas podem sofrer com atividades maliciosas originadas a partir de qualquer usuário conectado à rede.

É normal que o acesso à rede mundial de computadores ofereça a possibilidade de descoberta e exploração de vulnerabilidades de forma bem rápida, quase sempre mais do que a atualização das ferramentas de segurança e da emissão de correções pelos fabricantes de softwares. Então o número de incidentes de segurança cresce muito rapidamente juntamente com a quantidade de danos causados [3]. No entanto, proporcionalmente tem crescido também as pesquisas de novos mecanismos e técnicas para aumentar o nível de segurança. Políticas de acesso, uso de *firewalls*, sistemas de detecção de intrusão, sistemas de prevenção de intrusão, *honeypots*, entre outros, tem sido essas contra medidas.

Em segurança da informação é possível identificar os seguintes propriedades básicas, consideradas também os pilares para a segurança de redes [50]:

- **Confidencialidade:** Proteção dos dados contra divulgação não autorizada. A informação só deve estar disponível para aqueles devidamente autorizados;
- **Integridade:** Proteção dos dados contra modificação não autorizada. A informação não deve ser destruída ou comprometida e os sistemas devem ter um desempenho correto;
- **Disponibilidade:** Proteção de acesso aos recursos da rede para que estejam disponíveis quando requisitados pelas entidades autorizadas. Os serviços e recursos do sistema devem estar disponíveis sempre que forem necessários;
- **Autenticidade:** Proteção dos recursos da rede contra acesso não autorizado. É necessário a identificação dos elementos da transação;
- **Não-repúdio:** Proteção contra negação falsa de ações que um usuário ou entidade tenha feito. Não é possível negar a existência ou autoria de uma operação que criou, modificou ou destruiu uma informação;

- **Auditoria:** Implica no registro das ações realizadas no sistema, identificando os sujeitos e recursos envolvidos, as operações realizadas, seus horários, locais e outros dados relevantes;
- **Legalidade:** Proteção da informação no sentido de garantir a sua preservação, em conformidade com preceitos legais e também políticas de segurança da organização. Garante a legalidade jurídica da informação. Característica das informações que possuem valor legal dentro de um processo de comunicação, onde todos os ativos estão de acordo com as cláusulas contratuais pactuadas ou a legislação política institucional, nacional ou internacional vigentes.

Essas propriedades guiam para a determinação de um foco no qual uma determinada aplicação deve ser desenvolvida, a fim de atender os requisitos de segurança especificados pela própria necessidade do ambiente em questão. No entanto, não é apenas isto que dita o escopo de uma aplicação para segurança de redes, sendo importante delimitar a(s) fase(s) de proteção que ela irá atuar.

3.2 Fases para Proteção de Redes

É possível destacar quatro fases para proteger a rede contra ataques [97]: Prevenção, Detecção, Defesa e Forense, como pode ser visto na Figura 3.1. A **prevenção** compreende todos os métodos aplicados para evitar ataques, a fim de garantir a confidencialidade e integridade dos dados, com a utilização de controles de acesso aos recursos da rede. Isso inclui técnicas de autorização e autenticação (serviços de *login*), estabelecimento de confiança, bem como criptografia e filtragem de tráfego (uso de *firewall*). É importante ressaltar que prevenção só é possível para ataques conhecidos, mas há trabalhos sendo desenvolvidos no sentido de prever a ocorrência de ataques desconhecidos, como em [73] e [85].

3.2.1 Prevenção

Mecanismos de prevenção são considerados sistemas de defesa em primeira linha, ou seja, são responsáveis pela primeira etapa na segurança de uma rede de computadores. Normalmente essa defesa em primeira linha é feita na fase de projeto

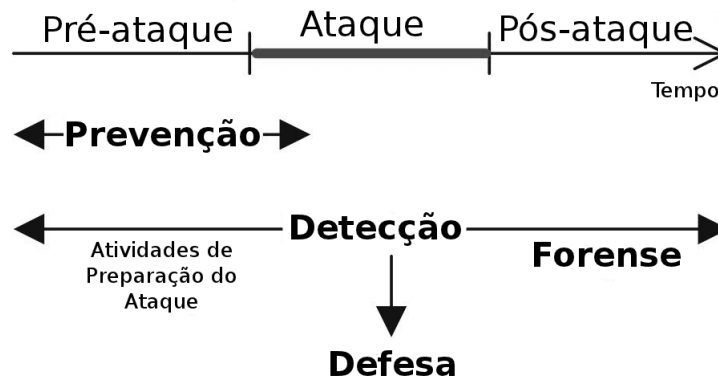


Figura 3.1: Quatro fases de proteção [97].

da rede, a fim de implantá-la de maneira segura e obter melhores resultados quando colocada em operação. Esses mecanismos são tipicamente implantados para controlar acesso a recursos e à informação que transita na rede.

Filtragem

Filtragem de pacotes é feita através de um conjunto de regras que analisam e filtram pacotes enviados por redes distintas de comunicação. Ela consiste no encaminhamento de pacotes, determinando se esses devem ser descartados ou permitidos, conforme o conteúdo de certos campos de seus cabeçalhos. Basicamente são inspecionados os endereços *Internet Protocol* (IP) de origem e destino, protocolo, portas de origem e destino e, em alguns casos, o estado da conexão.

Os filtros de pacotes normalmente podem ser implementados através de listas de controle de acesso - *Access Control Lists* (ACLs), um método utilizado para configurar e distribuir filtros em roteadores, e *firewall*, um dispositivo de segurança situado no limite da conexão com a Internet que examina os dados trafegados enquanto eles entram em uma das interfaces e aplica regras ao tráfego [92]. Portanto, é um dispositivo, ou conjunto de dispositivos, configurado para permitir, negar, encriptar, deciptar, ou encaminhar tráfego de dados entre computadores de diferentes domínios, baseado em um conjunto de regras ou critérios [61]. *Firewalls* podem ser implementados diretamente em hardwares ou através de softwares, ou ainda com a combinação de ambos (*firmware*).

Embora não exista um consenso sobre a classificação dos tipos de *firewall*, a mais usual considera a existência de três: filtro de pacotes, filtro de estados e filtros

de aplicação (*gateways*). O primeiro tipo, filtro de pacote, compara as informações do cabeçalho de cada pacote com as regras definidas para decidir qual ação tomar. A segunda, filtros de estado, mantêm registros do estado das conexões de rede que estão ativas. Assim, a filtragem é feita baseada na tabela de estados de conexões estabelecidas e não apenas no cabeçalho. O último tipo, *firewall* de aplicação, opera na camada de aplicação, vasculhando o conteúdo dos pacotes a procura de indícios de anomalias [42].

O sistema operacional Linux é considerado muito estável em relação a outros, pois não é comumente ameaçado por vírus de computador e tem o código aberto [95]. No *kernel* do Linux 2.4, foi introduzido o *firewall iptables* (comumente associado com *netfilter* [5]) que substituiu o *ipchains* dos *kernels* da série 2.2. Este *firewall* tem como vantagem ser muito estável, confiável e permitir muita flexibilidade na programação de regras pelo administrador do sistema [34].

O *iptables* é um *firewall* clássico em nível de pacotes (filtro de pacote). Ele é baseado no endereço/porta de origem/destino do pacote, prioridade, entre outros parâmetros. Ele funciona através da comparação de regras para saber se um pacote tem ou não permissão de acesso. Em *firewalls* mais restritivos, o pacote é bloqueado e registrado para que o administrador do sistema tenha conhecimento sobre o que está acontecendo em seu sistema. Ele também pode ser usado para muitas outras funcionalidades [34].

Cifragem ou Criptografia

É o uso de algoritmos matemáticos para transformar os dados em um formato que não seja prontamente decifrável. A transformação e subsequente recuperação dos dados depende de um algoritmo e uma ou mais chaves [87]. Ou seja, técnicas criptográficas permitem que um remetente disfarce os dados de modo que um intruso não consiga obter nenhuma informação dos dados interceptados. O destinatário deve estar habilitado a recuperar os dados originais a partir dos dados disfarçados [79].

3.2.2 Detecção

Se a prevenção falhar, detecção é a próxima fase a lidar com um incidente. **Detecção** é então o processo de descoberta de um ataque ou da preparação para sua realização, ou quaisquer outras atividades maliciosas, desde um *port scan* até indícios de quebra de senhas por técnica de força bruta. Isso normalmente é feito pela análise de dados capturados por um *sniffer*, sendo este a interceptação e registro de dados que trafegam pela rede.

IDS

Sistemas de detecção de intrusão - *Intrusion Detection Systems* (IDS) - são utilizados na fase de detecção. Eles realizam o monitoramento da rede, chamado de *Network Based Intrusion Detection System* (NIDS), ou de um dispositivo conectado a ela, bem conhecido como *Host Based Intrusion Detection System* (HIDS), com objetivo de encontrar a ocorrência de algum ataque ou atividade maliciosa. IDSs podem ser aplicados a diferentes arquiteturas de rede, como redes sem fio [25] e em dispositivos móveis [26]. Um IDS pode utilizar várias abordagens para identificar um ataque, sendo as mais conhecidas: Baseado em Assinatura (*Signature Based*) e Baseado em Anomalia (*Anomaly Based*) [56], sendo descritas em [47]:

- **Baseado em Assinatura:** Monitora as atividades da rede, ou de um dispositivo específico, procurando por ocorrências de alguma intrusão que é comparada com uma base de assinaturas, esta última contendo informações de características de ataques e atividades maliciosas;
- **Baseado em Anomalia:** Inicialmente é montado um perfil que representa o comportamento normal da rede ou *host*, fase conhecida como treinamento. Posteriormente o sistema entra em fase de monitoramento, no qual utiliza várias métricas para determinar se está havendo algum comportamento diferente do perfil inicial, caracterizando então uma intrusão.

Sistemas de Detecção de Intrusão baseados em assinatura provêm ótimos resultados na detecção de ataques específicos ou bem conhecidos. No entanto, eles não são capazes de detectar novos tipos de intrusões, mesmo que essas sejam variações

mínimas de ataques já conhecidos. Então o principal benefício em uma técnica de detecção baseada em anomalia é o seu potencial para detectar atividades maliciosas desconhecidas, ou "invisíveis". Porém, há como desvantagem a taxa de falsos positivos em sistemas baseados em anomalia, que geralmente é mais elevada do que os baseados em assinatura [33] [54].

Honeypots

Um outro mecanismo encontrado na fase de detecção é visto através das chamadas Metodologias de Decepção, definidas em [76] como a criação de um ambiente falso para enganar usuários mal intencionados. Elas são usadas quando aplicadas técnicas no qual o atacante interage com um recurso colocado como uma armadilha, propositalmente vulnerável, que emula os serviços ou sistemas que realmente deveriam ser alvos de sua ação.

Técnicas bastante utilizadas são com o uso de *honeypots*, definido por Spitzner [86] como um recurso de rede cuja função é de ser sondado, atacado ou comprometido. De acordo com Spitzner, o valor de um *honeypot* está diretamente ligado ao fato de que tal recurso não será utilizado por usuários legítimos ou para prover serviços para outros sistemas. Ou seja, nenhuma atividade envolvendo o *honeypot* será esperada, de modo que qualquer interação com ele pode ser considerada suspeita. Então são implementados de maneira que todo o tráfego destinado a eles seja anômalo ou malicioso, minimizando os falsos positivos.

Um *honeypot* poderá ser invadido, sendo uma máquina configurada a fim de obter informações sobre o invasor. A intenção é que o intruso ao realizar uma tentativa de invasão, na qual a rede possua um *honeypot* em funcionamento, tenha a sensação de está interagindo com uma máquina que exerce alguma funcionalidade e poderá conseguir algum recurso.

Existem mecanismos de segurança que funcionam de maneira ativa, diferente da utilização de *honeypot*, que é considerado passivo [65]. O problema da abordagem ativa é que o atacante toma a iniciativa para invadir de acordo com as vulnerabilidades específicas do mecanismo que ele identificou na rede. Dessa forma ele está sempre um passo à frente das tecnologias de segurança existentes.

Spitzner [86] classifica os *honeypots* quanto sua interação com o atacante, como segue:

- **Baixa Interatividade:** Provê serviços falsos, que aguardam conexões em uma determinada porta e responde ao atacante com respostas falsas. Os serviços podem ser emulados individualmente por ferramentas, sendo a mais conhecida o *Honeyd* [75], como também colocados em funcionamento com o uso de máquinas virtuais.
- **Alta Interatividade:** Uma máquina com sistema operacional e serviços reais, e não emulados, colocados para funcionar como *honeypot*. Este tipo oferece uma maior interação com o intruso, visto que o acesso é a um recurso real, com todas suas características. Porém, nesse tipo de *honeypots* é necessário uma atenção especial com a sua implantação, pois demanda uma maior estrutura de segurança para evitar que sejam comprometidos, podendo passarem a ser utilizados como ponte para a realização de novos ataques.

Na Tabela 3.1 é mostrado um comparativo entre *honeypots* de baixa e alta interatividade:

Tipo	Baixa Interatividade	Alta Interatividade
<i>Emulação</i>	Emulam sistemas e serviços	Executam sistemas e serviços reais
<i>Implementação</i>	Simples e de fácil gerenciamento	Cuidados devem ser tomados na instalação para que não sejam comprometidos
<i>Controle</i>	Atacante não tem controle, por se tratar de uma emulação	Atacante pode ter controle total, por se tratar de um serviço real
<i>Limitações</i>	Captura de tráfego e <i>malware</i>	Captura de mais informações
<i>Eficiência</i>	Menor facilidade em iludir atacantes	Dificuldade em identificar na rede, pois são confundidos com serviços de produção

Tabela 3.1: Comparação entre os tipos de *honeypots*.

Um outro conceito é o de *honeynet* relacionado a metodologias de decepção, sendo esta um conjunto de *honeypots* juntamente com um *firewall* para limitação e registro de tráfego [86]. Normalmente ela é virtualizada em uma única máquina, sendo os endereços IP, serviços de rede e sistemas operacionais emulados. É uma rede projetada especialmente para ser atacada.

Segundo Stallings [87] os *honeypots* são projetados para:

- Desviar um atacante do acesso a sistemas críticos;
- Coletar informações sobre a atividade do atacante;
- Encorajar o atacante a permanecer no sistema por tempo suficiente para que os administradores respondam.

Pode-se citar, de acordo com Spitzner [86], algumas vantagens na utilização de *honeypots*:

- Como ele é isolado, o registro de informações para análise é pequeno;
- Acaba com falsos positivos gerados por IDS;
- Exigência de recursos mínimos, pois podem ser virtualizados;
- Captura de tráfego criptografado;
- Descoberta de novas tecnologias utilizadas por usuários mal intencionados.

Como também desvantagens:

- Visão limitada de tráfego, pois há a possibilidade do atacante reconhecer o *honeypot* e não interagir com ele;
- Risco de ser invadido e utilizado para prejudicar outros sistemas, se tornando comprometido;
- Ausência de tráfego, quando a tecnologia não é bem aplicada, implicando em gastos sem retorno, já que nada foi monitorado.

Muitos trabalhos são realizados no sentido de explorar os benefícios oferecidos pelos recursos dos *honeypots* em segurança de redes. Alguns trabalhos realizam aplicações em outras tecnologias como: redes sem fio [15], redes de sensores [62], redes veiculares [93] e em dispositivos móveis [96] [64]. Também os utilizam como recursos educacionais, no ensino e pesquisa, na geração dados estatísticos sobre tráfego malicioso em redes e na Internet [57] e no combate a *spams* [37].

3.2.3 Defesa

Se um tráfego malicioso é detectado, então é iniciado o processo ou fase de **defesa**. Para que isso ocorra é necessário que o sistema implemente essas duas fases (detecção e defesa), integrando mais de uma aplicação de segurança.

IPS

Um exemplo de aplicação utilizada nesta fase é o Sistema de Prevenção de Intrusão - *Intrusion Prevention System* (IPS), sendo este uma extensão dos IDSs, normalmente implementado com a utilização de *plugins*. Ele gera alguma resposta a fim de neutralizar o ataque, podendo integrar, por exemplo, um IDS a um *firewall* ou enviar uma mensagem de *Short Message Service* (SMS) ou e-mail ao administrador da rede informando a ocorrência de atividade(s) maliciosa(s).

Dessa forma, surge os Sistemas de Detecção de Intrusão Colaborativos - *Collaborative Intrusion Detection System* (CIDS), um IDS, ou IPS, com capacidade de analisar evidências de múltiplas redes simultaneamente. Um CIDS tem a função de detectar ataques coordenados em estágio inicial, antes que tenham causado danos maiores à Internet [33].

Softwares anti-*

Programas desenvolvidos para detectar e eliminar potenciais ameaças aos computadores e redes como, por exemplo, antivírus, *antispyware*, *antiphishing* e *antispam*. Antivírus são programas que detectam e removem vírus de computador. Já os *antispywares* são usados no combate a programas e códigos espiões como *spyware*, *adware*, *keyloggers*. Softwares *antiphishing* visam bloquear possíveis tentativas de fraude através de sites Web ou mensagens de correio eletrônico. Normalmente, este tipo de solução é apresentado na forma de barras de tarefas integradas com navegadores Web ou clientes de correio eletrônico. As soluções *antispam* também se enquadram nesta categoria. Basicamente, a detecção de *spam* é baseada na filtragem de mensagens não solicitadas através dos campos do cabeçalho ou do conteúdo da mensagem [42].

3.2.4 Forense

Em alguns casos quando as fases anteriores de prevenção e detecção falham e o ataque foi bem sucedido, se faz necessário uma análise de todos os logs, no sentido de aprender como os métodos utilizados nos processos de detecção e defesa podem ser melhorados para futuros incidentes. Sendo esta a fase chamada de **forense**, que tem como objetivo realizar uma investigação para descobrir detalhes específicos de ataques, apresentando resultados que contribuam de alguma maneira para a melhoria da proteção da rede. Outro objetivo desta fase é fornecer provas suficientes para permitir que o autor do incidente de segurança seja processado. Para Pilli ES et al., em [71], as técnicas de forense de rede fornecem recursos para os investigadores rastreamos os atacantes.

Perícia Digital

Para Palmer [67] Informática Forense é definida como o uso de métodos cientificamente estabelecidos e comprovados para a preservação, coleta, validação, identificação, análise, interpretação, documentação e apresentação da evidência derivada de fontes digitais para o propósito de facilitar ou promover a reconstrução de eventos que causem a perturbação de operações planejadas. A perícia digital tem uma natureza investigativa em crimes relacionados a computadores [43].

Diante destas fases é possível caracterizar a atuação das aplicações de segurança de redes. Por sua vez, muitas das aplicações utilizadas atualmente possuem problemas. Junto a isso, os ataques à redes de computadores vem se tornando cada vez mais sofisticados. Esses problemas serão detalhados a seguir.

3.3 Problemas Enfrentados

Nesta seção será descrito em duas classificações os problemas em segurança de redes, de modo a facilitar a compreensão do nível de complexidade em que se encontram. São eles: problemas enfrentados pelos atuais sistemas de segurança e a robustez dos ataques a sistemas computacionais.

3.3.1 Problemas em Sistemas de Segurança

Zseby et al., em [97], afirmam que a segurança de redes necessita crucialmente de uma maior atenção por parte do administrador e quase sempre de mais esforços e custos, pois novos protocolos e aplicações introduzem novas vulnerabilidades. Infelizmente, os mecanismos utilizados para aumentar o nível de segurança atualmente possuem alguns problemas, a saber:

- **Estrutura sólida:** Sistemas desenvolvidos para segurança possuem pouca flexibilidade, pois são desenvolvidos apenas para situações isoladas. Não possuem as características de um sistema aberto. Também não são capazes de se integrar com outros mecanismos de segurança existentes no ambiente;
- **Habilidade de defesa baixa:** O escopo das atuais aplicações de segurança é limitado. São desenvolvidas apenas para atender alguma das fases de proteção, sem ter a habilidade para prover um mecanismo integrado que forneça maior capacidade de segurança;
- **Sem autoadaptação:** Aplicações não tem a capacidade de mudar componentes internos para se adaptarem às necessidades do ambiente em que se encontram a fim de conseguir provê segurança à rede. Todo um processo de reconfiguração manual é preciso para manter a segurança;
- **Sem autoaprendizagem:** Aplicações não possuem a habilidade para aprender com o tempo que se encontram em funcionamento, necessitando de intervenção para se atualizarem. Não conseguem aprender com suas próprias ações, verificando os resultados destas que foram providos à rede, para uma análise e melhorias futuras;
- **Sem auto-evolução:** Além de não possuírem capacidade para aprenderem, ainda não implementam meios para que novos conhecimentos sejam agregados às técnicas de defesa.

Para Atay e Masera, em [18], todos os métodos de análise de ameaças, vulnerabilidades e riscos necessitam atualizarem continuamente os seus conhecimentos sobre as novas fraquezas encontradas nos ativos da rede. Isso servirá para identificar como estas fraquezas podem ser exploradas, para posteriormente

definir e aplicar as contra-medidas necessárias. Esse é um ciclo contínuo, pois novas avaliações serão necessárias ao longo do tempo.

No entanto, sabe-se que informações sobre novos ataques não são imediatamente divulgadas, pelos fabricantes ou pela comunidade que desenvolve o software, devido à sua sensibilidade. Isso ocorre devido essas informações poderem ser utilizadas para explorar mais ainda as vulnerabilidades, visto que usuários mal intencionados podem obtê-las. Elas são publicadas então logo após os fabricantes liberarem as correções. Os métodos de análise de riscos citado por [18] devem refazer a avaliação de segurança dos sistemas levando em consideração informações de novos ataques quando forem divulgadas, ou com a utilização de técnicas de inteligência para detecção antes mesmo da divulgação.

Diante desta situação, Wang et al. [52] afirmam que a direção correta para o desenvolvimento de aplicações de defesa e segurança é com a adoção de duas características: a integração e inteligência. Integração para possibilitar o gerenciamento de vários recursos de proteção em um ambiente de rede distribuído e a inteligência para prover adaptação ao ambiente, aumentar a eficiência de proteção baseado em seu conhecimento e no que adquiriu e, por fim, alcançar um equilíbrio entre a aplicação de segurança e o ambiente de rede.

Os problemas dos Antivírus

Em [31] foram listados os problemas mais recorrentes em softwares antivírus, como segue:

- Surgimento frequente e crescente de variantes de *malware* previamente identificados, cujas ações modificadas visam evadir a detecção. Na tentativa de evoluir e conseguir defender-se dessas variantes, podem ocorrer falsos-positivos. Esses, por sua vez, são uma outra dificuldade no processo de defesa, pois podem identificar erroneamente uma aplicação inofensiva do usuário como sendo maliciosa e, conseqüentemente, removê-la, colocá-la em quarentena (bloqueio) ou restringir de algum modo;
- Muitos exemplares de *malware* possuem mecanismos próprios de defesa cujas ações variam entre desabilitar as proteções existentes no sistema operacional

alvo, verificar se o exemplar está sob análise (o que pode causar uma modificação em seu comportamento em razão disto), e disfarçar-se de programas do sistema, inclusive de falsos antivírus;

- A comunidade de desenvolvedores de antivírus ainda não possui padronização para a classificação dos *malware* identificados por suas ferramentas. Isso faz com que o processo de resposta a incidentes de segurança envolvendo código malicioso seja prejudicado, tornando-o ineficiente em determinados casos;
- Para permitir a remoção do código malicioso e as atividades derivadas da infecção do sistema comprometido é necessário, em muitos casos, uma intervenção manual do usuário.

3.3.2 Robustez dos Ataques a Sistemas Computacionais

Aliado aos problemas enfrentados pelos atuais sistemas de segurança, os ataques a sistemas computacionais estão se tornando cada vez mais sofisticados, imprevisíveis, frequentes e com uma maior quantidade de origens [18]. Como exemplo destes ataques e também atividades maliciosas, serão destacados alguns a seguir.

Disseminação de Códigos Maliciosos

A disseminação de código malicioso através da Internet tem sido a maior ameaça atual à segurança de sistemas de informação. Código Malicioso ou *Malware* é um termo genérico que abrange todos os tipos de programas desenvolvidos especificamente para executar ações maliciosas em um computador. Esta denominação é comumente empregada a um conjunto de aplicações também denominadas individualmente como vírus, *worms*, *keyloggers*, *trojans*, *backdoors* e outros tipos de programas com a intenção de comprometer um sistema [31].

Uma das maneiras mais utilizadas para o comprometimento de sistemas é através da propagação autônoma de *malwares*. A busca por máquinas vulneráveis com o intuito de explorar vulnerabilidades e comprometer o sistema é característica de *malwares* como *worms* e *bots*. Uma forma de entender as características dos diferentes tipos de *malware* é realizar a análise de tráfego malicioso [42].

Spam e Phishing

O termo *spam* é utilizado para referenciar o recebimento de uma mensagem não solicitada, que geralmente tem o caráter de fazer propaganda de algum produto ou assunto não desejado. A palavra *spammer* é o termo utilizado para definir quem envia esse tipo de mensagem, em cuja elaboração geralmente são utilizados softwares especiais automatizados para coletar bases de e-mails através de listas de discussão, caixas postais de grupos, ou exploração, através de listas adquiridas, por empresas especializadas nesse tipo de marketing. Existem diversas soluções no mercado que buscam conter o recebimento de *spams*, ferramentas estas que basicamente utilizam filtros contendo *blacklists* de *proxys* e *relays* abertos, que são utilizados para o envio das mensagens [38] [28].

O termo *phishing* refere-se ao método pelo qual iscas (e-mails) são usadas para pescar informações de usuários da Internet, constituindo-se um ataque que tem como objetivo efetuar algum ilícito através do envio de mensagem não solicitada, realizados por *phishers*. Em ataques usando engenharia social, o *phisher* envia mensagens eletrônicas, normalmente e-mails ou mensagens instantâneas, alegando ser uma entidade legítima (bancos, SPC/SERASA, receita federal, entre outros) ou uma pessoa conhecida. O conteúdo dessas mensagens apresenta formulários para o preenchimento e o envio de dados pessoais e financeiros, links para baixar e abrir/executar programas ou links para uma página Web construída especificamente para que o destinatário da mensagem forneça dados pessoais e financeiros. Em ataques que utilizam artifícios técnicos, o criminoso instala no computador da vítima software maliciosos, como *trojans*, *keylogger* e *spywares*, para roubar dados sigilosos [28].

Botnets

De acordo com Rajab et al., em [12], *botnet* é uma rede de computadores comprometidos controlados remotamente por um operador humano chamado de *botmaster*. Um *bot* é uma aplicação residente em um nó integrante de uma *botnet* com capacidade de se autopropagar infectando outros *hosts* vulneráveis [24].

Botnets são plataformas de computação distribuída predominantemente usadas para atividades ilegais com o objetivo de lançamentos de ataques de negação

de serviço distribuído - *Distributed Denial of Service* (DDoS), envio de *spam*, *trojan* e e-mails *phishing*, distribuição ilegal de mídias e softwares piratas, roubo de informações e de recursos computacionais, realização de fraudes e falsidade de identidade [30].

Ataques de Último Dia

Um ataque de último dia, do inglês *Zero-day Attack* ou *0-Day Attack*, também conhecido como ataque de dia zero ou zero hora, aproveita vulnerabilidades que atualmente não tem uma solução ou ainda não são conhecidas [73] [85]. Normalmente, é descoberto um *bug* ou um problema em um software depois que ele foi liberado, sendo então em seguida oferecida uma correção destinada ao problema original. Um ataque de último dia vai aproveitar esse problema antes da correção ser criada. Na maioria dos casos, esse tipo de ataque vai tirar proveito de um problema não conhecido pelos criadores do software e seus usuários.

É importante ressaltar que nem todos os ataques de último dia realmente acontecem antes dos produtores de software estarem cientes da vulnerabilidade. Em alguns casos eles conhecem a vulnerabilidade, mas o desenvolvimento da correção pode demandar um tempo elevado, em frente a complexidade do problema enfrentado. Neste sentido, proteção de último dia é a capacidade de um sistema de segurança fornecer proteção contra ataques de último dia.

3.4 Aplicações de Virtualização em Segurança

Com o objetivo de contornar problemas de compatibilidade de hardware, foram desenvolvidas várias tecnologias, o que originou o termo virtualização. Usando os serviços oferecidos por uma determinada interface de sistema é possível construir uma camada de software que ofereça aos demais componentes uma outra interface, a chamada camada de virtualização construída em software [53]. Essa camada de software permite o acoplamento entre interfaces distintas, de forma que um programa desenvolvido para a plataforma A possa executar sobre uma plataforma distinta B.

Através dos serviços oferecidos por uma determinada interface de sistema, a camada de virtualização constrói outra interface de mesmo nível, de acordo com as necessidades dos componentes de sistema que farão uso dela. A nova interface

de sistema, vista através dessa camada de virtualização, é denominada máquina virtual [83]. A camada de virtualização em si é denominada hipervisor ou monitor de máquina virtual, como visto na Figura 3.2.

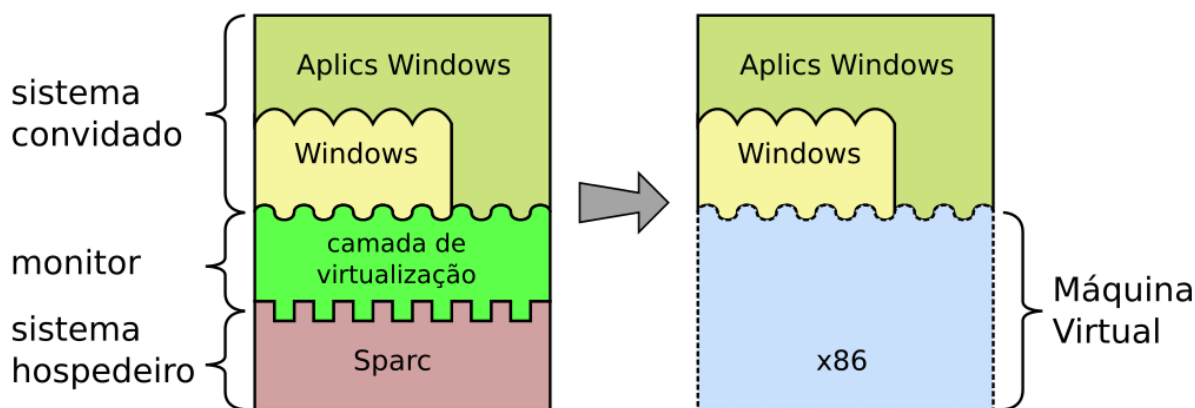


Figura 3.2: Uma máquina virtual [82] [53].

Existem várias formas de implementar a virtualização, as principais são apresentadas [53], também vistas na Figura 3.3:

- **Virtualização completa:** Um sistema operacional convidado e suas aplicações, desenvolvidas para uma plataforma de hardware A, são executadas sobre uma plataforma de hardware distinta B;
- **Emulação do sistema operacional:** As aplicações de um sistema operacional X são executadas sobre outro sistema operacional Y, na mesma plataforma de hardware;
- **Tradução dinâmica:** As instruções de máquina das aplicações são traduzidas durante a execução em outras instruções mais eficientes para a mesma plataforma;
- **Replicação de hardware:** Várias instâncias virtuais de um mesmo hardware real são criadas, cada uma executando seu próprio sistema operacional convidado e suas respectivas aplicações.

A evolução da tecnologia de máquinas virtuais tem permitido sua ampla adoção em sistemas de produção. Vários trabalhos de pesquisa e de desenvolvimento nos últimos anos comprovaram a eficácia da utilização de máquinas virtuais no campo da segurança de sistemas. Algumas das propriedades conceituais da virtualização

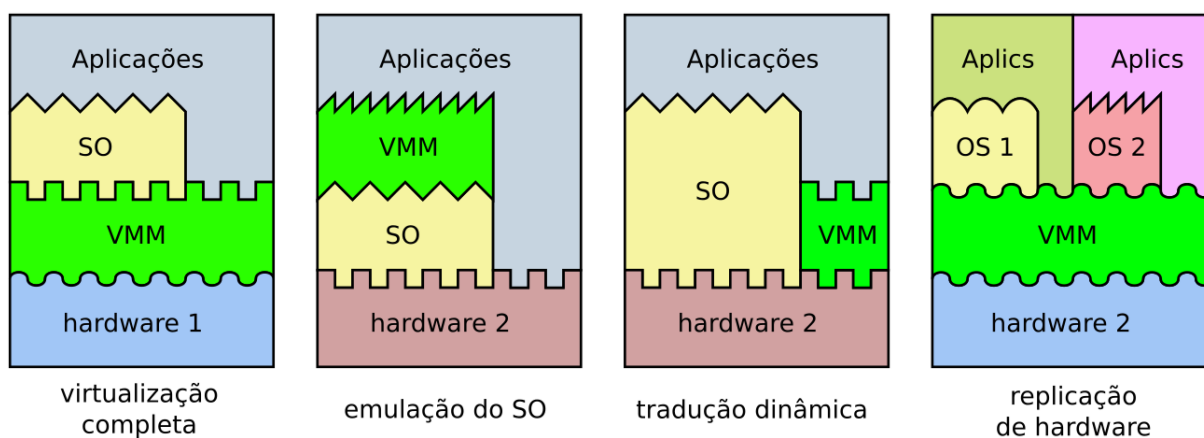


Figura 3.3: Forma de implementar virtualização [82] [53].

podem ser úteis para se obter as propriedades básicas de segurança da informação (Seção 3.1), tais como [53]:

- **Isolamento:** Ao manter os ambientes virtuais isolados entre si e do sistema real subjacente, o hipervisor provê a confidencialidade de dados entre os sistemas convidados. Adicionalmente, como os dados presentes em uma máquina virtual só podem ser acessados pelas respectivas aplicações convidadas, sua integridade é preservada. Além disso, o isolamento permite a contenção de erros de software acidentais ou intencionais no âmbito da máquina virtual, o que permite melhorar a disponibilidade dos sistemas;
- **Controle de recursos:** Como o hipervisor intermedeia os acessos do sistema convidado ao hardware, é possível implementar mecanismos para verificar a consistência desses acessos e de seus resultados, aumentando a integridade do sistema convidado; da mesma forma, é possível acompanhar e registrar as atividades do sistema convidado, para fins de auditoria;
- **Inspeção:** A visão privilegiada do hipervisor sobre o estado interno do sistema convidado permite extrair informações deste para o sistema hospedeiro, permitindo implementar externamente mecanismos de verificação de integridade do ambiente convidado, como antivírus e detectores de intrusão; além disso, a capacidade de inspeção do sistema convidado, aliada ao isolamento provido pelo hipervisor, torna as máquinas virtuais excelentes para o estudo de aplicações maliciosas;

- **Encapsulamento:** A possibilidade de salvar/restaurar o estado do sistema convidado torna viável a implementação de mecanismos de *rollback* úteis no caso de quebra da integridade do sistema convidado; da mesma forma, a migração de máquinas virtuais é uma solução viável para o problema da disponibilidade.

3.5 Conclusão

Este capítulo apresentou uma introdução a segurança de redes, mostrando as fases em que uma rede de computadores pode ser protegida. Discorreu sobre as ferramentas e técnicas mais utilizadas em segurança de redes, classificando-as de acordo com sua fase de atuação.

Foram descritos também os atuais problemas que a segurança de redes possuem. Para um melhor entendimento, eles foram apresentados em duas classificações: primeiramente os problemas enfrentados pelos atuais sistemas de segurança e, em seguida, o alto nível de robustez em que se encontram os ataques a sistemas computacionais. Ao final deste capítulo foram vistos os conceitos fundamentais sobre virtualização e suas principais possibilidades de uso em segurança da informação.

4 Trabalhos Relacionados

Para dar uma visão com mais alto grau de realismo da aplicação dos conceitos de CA ao contexto da segurança de redes, são vistos os seguintes tópicos neste capítulo:

- Trabalhos relacionados que utilizam a CA como base para o desenvolvimento de suas propostas;
- Trabalhos relacionados que implementam alguma das propriedades oferecidas pela CA;
- Caracterização da proposta.

4.1 Sistemas de Segurança baseados em Computação Autônoma

ISDS

Em [52] foi desenvolvido o *Intelligent Security Defensive Software* (ISDS), um modelo de software de segurança baseado em CA. A estratégia do ISDS é fazer o processo de construção do software de segurança fornecendo a ele inteligência. Em outras palavras, construir um software utilizando o modelo ISDS é fazer com que seus componentes se modifiquem dinamicamente de acordo com a situação de segurança atual da rede. Para isto, o ISDS fornece consciência do contexto.

O ISDS é um modelo de sistema de defesa distribuído caracterizado por ser flexível e autoadaptativo. Os domínios nos quais ele pode ser aplicado são principalmente no gerenciamento de segurança em redes locais e na gerência de segurança da informação na Internet. A ideia do modelo é que o sistema possa analisar as informações do ambiente e ajustar sua estrutura e maneira de agir dinamicamente. Ele consiste de alguns componentes básicos, são eles: componente de comando,

componente de execução, componente sensor, componente de políticas, componente de equipamento e outros auxiliares como visto na Figura 4.1.

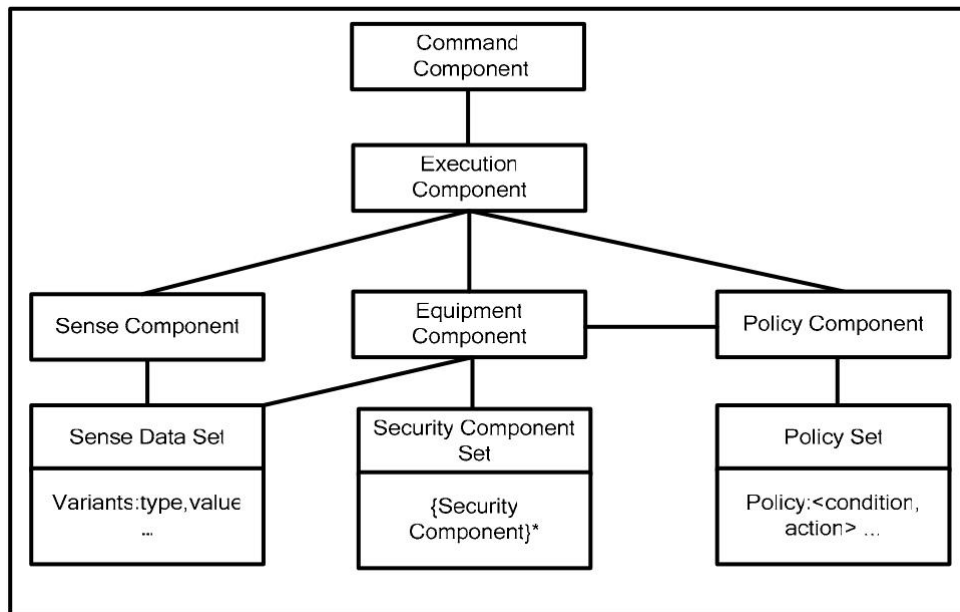


Figura 4.1: *Framework* conceitual do ISDS [52].

O componente de comando (*Command Component*) é a principal parte do ISDS, tendo como funções: a ativação dos componentes de políticas (*Policy*), sensor (*Sense*) e equipamento (*Equipment*), a execução da tomada de decisão baseada no componente sensor, o gerenciamento do conjunto de componentes de segurança e política, atualização de novos componentes de segurança e políticas, verificando e resolvendo conflitos de políticas. Já o componente de execução (*Execution Component*) trabalha como um canal de comunicação entre as entidades de segurança e o componente de comando. Ele se encarrega de filtrar, tratar e transmitir a mensagem para todos os outros componentes, os fazendo trabalharem corretamente.

O componente de políticas se encarrega principalmente em fazer a manutenção do conjunto de políticas, a tomada de decisão das políticas adequadas e também a passagem do resultado da decisão. As informações do resultado das decisões tomadas serão passadas para o componente de equipamento que então irá executar alguma ação. Por fim, o componente sensor coleta informações do ambiente e as envia para o componente de comando, levando em consideração o custo do sensoriamento de dois modos: emergência e *timing*, sendo o primeiro com maior prioridade em relação ao segundo.

Autonomic Security and Self-Protection based on Feature-Recognition with Virtual Neurons

No trabalho desenvolvido em [27] foi apresentado um mecanismo de segurança autônomo baseado em neurônios virtuais e no reconhecimento de características. Sua abordagem trabalha para detectar automaticamente vários problemas de segurança que atualmente são difíceis de realizar defesa. Através de simulação e diferentes estudos de caso, resultados mostram que esta solução é viável.

Os neurônios virtuais são desenvolvidos de forma análoga aos neurônios dos animais, como visto na Figura 4.2. A ideia é que eles sejam distribuídos de maneira a perceber modificações no ambiente e reagir a estímulos. Esse neurônio virtual é na realidade um simples software com capacidade de ciência de contexto, a fim de analisar informações do contexto e estar ciente de eventuais surgimentos de ataques.

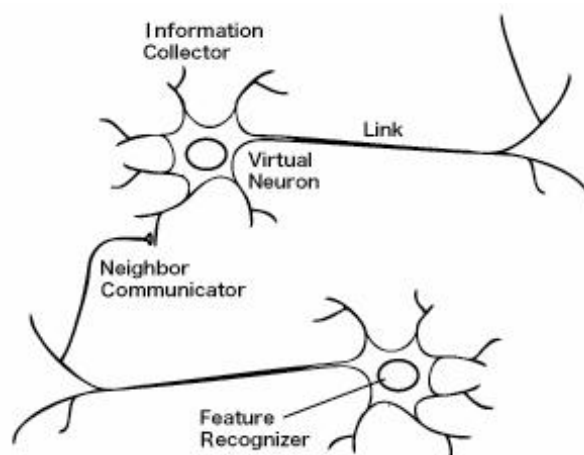


Figura 4.2: Neurônio Virtual [27].

No neurônio virtual, o componente *Information Collector* captura várias informações de contexto, tais como uso de memória e processador, *status* dos processos e informações de tráfego da rede. Os neurônios que se conectam diretamente são considerados vizinhos, comunicando-se através do *Neighbor Communicator*. Um outro componente é utilizado, chamado de *Feature Recognizer*, sua função é de identificar atividades maliciosas, sendo baseado no conhecimento sobre informações que caracterizam um ataque (baseado em assinaturas). Essas informações são passadas para o *Feature Recognizer* pelo *Information Collector* do próprio neurônio e dos seus vizinhos, através do *Neighbor Communicator*.

Os neurônios virtuais podem ser facilmente distribuídos, pois o pacote de instalação é compacto (tamanho pequeno), eles exigem poucos recursos computacionais e são de fácil instalação nos *hosts*. Eles são distribuídos em uma arquitetura hierárquica e Par-a-par - *Peer-to-peer* (P2P), como visto na Figura 4.3. A estrutura P2P opera baseada nas relações com neurônios vizinhos, tendo a comunicação entre eles através de passagem de mensagens. A arquitetura hierárquica é utilizada para aumentar a eficiência na propagação das mensagens por grupo de neurônios, chamados de células. Em cada célula um neurônio é eleito como neurônio-chefe (*Head Neuron*), o algoritmo de eleição leva em consideração os recursos computacionais e velocidade de comunicação do *host*.

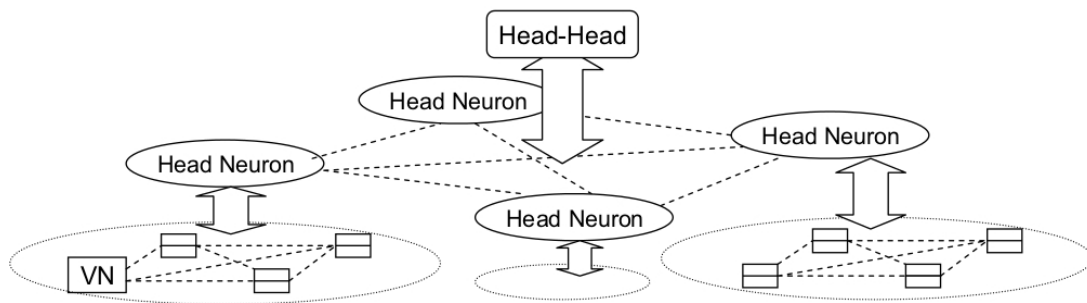


Figura 4.3: Estrutura de organização hierárquica e P2P dos neurônios virtuais [27].

Na organização hierárquica, um neurônio-chefe é eleito como o chefe de alto nível (*Head-Head*) para outros neurônios-chefes. Este procedimento é repetido até existir um único neurônio-chefe de maior nível sobre os demais. Se ocorrer alguma falha em algum neurônio-chefe de uma célula, um outro neurônio desta mesma célula irá assumir o papel de neurônio-chefe, através de uma nova eleição. Nessa organização hierárquica as mensagens são entregues de maneira mais eficiente.

O mecanismo de segurança empregado por este sistema distribuído é feito na detecção de mensagens com dados ou mensagens ilegais. Primeiramente as origens dos ataques são rastreadas para identificar e localizar o atacante. Depois o tráfego ilegal originado por ele é bloqueado, ou seja, todos os neurônios irão receber mensagens para descartarem tráfego que tenha como origem um usuário malicioso identificado. A reconfiguração de recursos é feita neste momento para realização da defesa pelo sistema. Para realizar a detecção e posterior defesa foi desenvolvido um mecanismo (característica ou assinatura) para cada tipo de ataque, dentre esses:

Eavesdropping, Replay, Masquerading, Spoofing e Negação de Serviço - *Denial of Service* (DoS).

Self-Configuration of Network Security

Este trabalho [22] apresenta uma abordagem de autoconfiguração para controlar e gerenciar os mecanismos de segurança em redes de grande porte. Nela o sistema automaticamente configura os mecanismos de segurança e modifica as configurações dos recursos e suas políticas em tempo de execução. Para mostrar sua viabilidade foi implementado o *Autonomic Network Defense System* (AND). AND é um sistema que pode detectar ataques de rede conhecidos ou desconhecidos (ataques de último dia) e proativamente prevenir ou minimizar impactos causados em serviços da rede.

O AND foi desenvolvido sobre o *framework* AUTONOMIA [29], sendo uma extensão focada em estratégias e mecanismos para detectar e proteger-se de ataques de rede. O AUTONOMIA possui dois módulos de software, são eles: *Component Management Interface* (CMI) e *Component Runtime Manager* (CRM). O CMI é utilizado para especificar as configurações e políticas associadas com cada componente (de hardware ou software) e o CRM gerencia as operações dos componentes usando as políticas definidas no CMI.

As principais extensões do AND são os módulos de Monitoramento Online (*Online Monitoring*), Seleção de Característica (*Feature Selection*), Análise de Anomalia (*Anomaly Analysis*) e o de Ação (*Action Module*), como visto na Figura 4.4. É possível observar que essa arquitetura é baseada no MAPE-K.

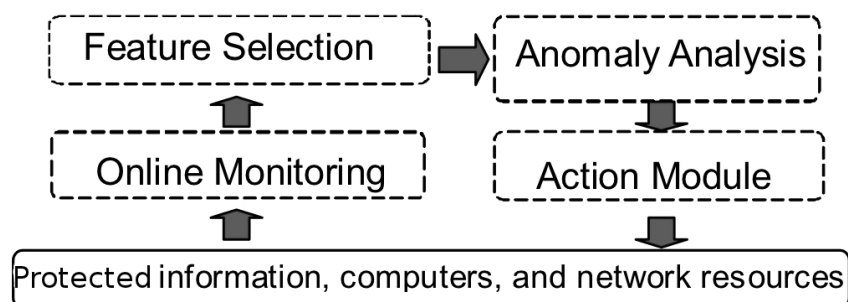


Figura 4.4: Arquitetura do AND [22].

O monitoramento online coleta os dados trafegados na rede e operações realizadas em computadores através de ferramentas específicas e arquivos de log gerados por sistemas operacionais. O modelo de seleção de característica irá filtrar os dados monitorados a fim de encontrar informações ou características relevantes para serem passadas para o módulo de análise de anomalias. Neste é executada uma função para quantificar se existe um desvio do perfil padrão da rede ou de algum sistema, caso haja, é considerado uma anomalia e o módulo de ação irá executar ações adequadas. Este último irá, resumidamente, restringir o acesso aos recursos atacados e posteriormente tentar desinstalar códigos maliciosos que estão executando em computadores comprometidos da rede.

Qualidade de Proteção

Este trabalho foi iniciado em [77] e teve como resultado um novo termo em [41] chamado de Qualidade de Proteção - *Quality of Protection* (QoP). Ele é um *framework* proativo para defesa de redes que pode ser integrado com os já existentes protocolos de Qualidade de Serviço - *Quality of Service* (QoS). O objetivo é provê serviços diferenciados para fluxos de tráfegos de acordo com seu grau de anormalidade.

Para isso foi criado uma nova métrica chamada de Distância de Anormalidade (DA), como visto na Figura 4.5, que pode ser usada para classificar o tráfego em normal, incerto (tráfego suspeito) e anormal (tráfego malicioso). Existe uma função *Delta* que mostra o quanto mais próximo o tráfego está de normal ou anormal, podendo ser classificado então como provavelmente normal ou provavelmente anormal.

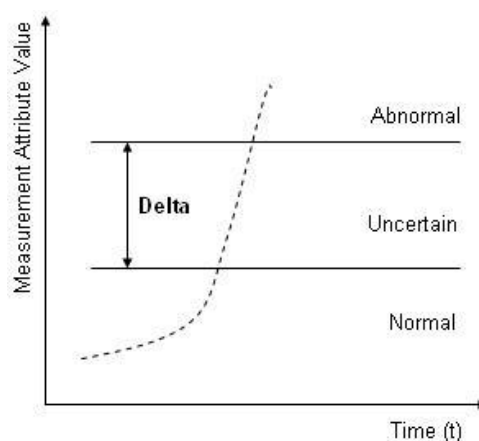


Figura 4.5: Distância de Anormalidade [77].

A ideia é que a métrica DA seja usada em conjunto com protocolos de QoS para aumentar a prioridade de tráfegos considerados normais e diminuir a prioridade de tráfegos anormais. Testes foram realizados com ataques de DDoS e propagação de *Worms*. Com isso foi possível demonstrar o quanto a abordagem proposta pode detectar e reduzir o impacto causado pelos ataques.

4.2 Propriedades de Computação Autônômica em Sistemas de Segurança

Alguns trabalhos se caracterizam por fornecer alguma(s) das propriedades autônômicas de maneira isolada, mesmo não tendo como base a CA. Ou seja, foram desenvolvidos sem seguir os conceitos da CA, mas provem algum mecanismo que se qualifica dentre alguma das propriedades autônômicas. Sistemas como estes não são considerados autônômicos.

Foi desenvolvido em [94] um esquema de segurança para atualização de regras de *firewall* baseado no tráfego de uma *honeynet* (autoconfiguração e autoaprendizagem). No esquema há um módulo de análise de dados que oportunamente descobre novos ataques. Este módulo analisa o tráfego gerado pelos logs da *honeynet* e com a utilização de mineração de dados ele cria dinamicamente novas regras de *firewall* passando-as para o módulo de aprendizagem de regras, que as filtra eliminando incoerências entre as regras e, por fim, ele as aplica. Dessa maneira o *firewall* continua enriquecendo suas estratégias melhorando sua capacidade para se defender de novos ataques, inclusive ataques de último dia.

A ferramenta *Honeycomb* [51] tem como objetivo automatizar a geração de assinaturas de ataques para sistemas de detecção de intrusão a partir de logs gerados por *honeypots* (autoaprendizagem). Na verdade esta ferramenta trata-se de um *plugin* a ser utilizado junto ao *honeyd* [75], que é um *framework* de *honeypots* de baixa interatividade [86]. A implementação do *Honeycomb* gera assinaturas no formato *Bro* [70] e *Snort* [78]. Sua intenção é ser utilizado para criar as assinaturas de ataques em tempo de execução (autoconfiguração), atividade que normalmente é realizada manualmente por especialistas de segurança após o registro das atividades maliciosas nos logs.

No caso do *SweetBait* [72], foi desenvolvido um sistema automatizado de proteção que utiliza *honeypots* de baixa e alta interatividade para reconhecer e capturar tráfego malicioso. Com base nos logs gerados, após descartar padrões da lista branca, ele automaticamente cria assinaturas de ataques (autoaprendizagem), componente implementado utilizando o *Honeycomb*. Para provê um baixo tempo de resposta ao ataque ele distribui imediatamente assinaturas para IDSs (autoconfiguração), após sua geração. Paralelamente a isto, as assinaturas são refinadas continuamente a fim de aumentar sua precisão e diminuir falsos positivos (auto-otimização). Este trabalho é estendido e surge então o *Argos* [73], um sistema de resposta automatizado, que emprega uma característica mais ampla, a propriedade de autoproteção, por reagir proativamente contra ataques.

4.3 Caracterização da Proposta

Esta seção apresenta a caracterização da proposta deste trabalho. O mecanismo autônomo para proteção de redes desenvolvido neste trabalho pode ser dividido em três tópicos, de maneira a representar mais claramente o escopo da proposta e mostrar o que ela trouxe de acréscimo para a área de pesquisa e as suas diferenças em relação aos trabalhos que foram desenvolvidos anteriormente. Esses tópicos são listados a seguir:

1. Um *framework* autônomo, no qual é organizado seguindo o modelo MAPE-K;
2. Um ciclo autônomo que tem a funcionalidade de gerar regras de *firewall* baseadas em logs de *honeypots*;
3. Um ciclo autônomo responsável por manipular dinamicamente *honeypots* virtuais que são considerados comprometidos.

Inicialmente, a proposta do *framework* é inspirada em outros trabalhos desenvolvidos em [49] [80] [29], que visam também fornecer comportamento autônomo para sistemas computacionais. Diferentemente dos demais, a nossa proposta foca na simplicidade e baseia-se no modelo MAPE-K, tentando representar fielmente todos seus componentes com suas funcionalidades específicas

e responsabilidades. Porém, nosso *framework* é criado com o objetivo de ser aplicado em ambientes de redes, para cooperação entre sistemas de segurança.

Com a utilização do *framework*, o primeiro ciclo autônomo é implementado a fim de melhor utilizar os logs gerados por *honeypots*, inspirado nos trabalhos desenvolvidos em [85] [90] [16] e principalmente em [20]. Ambos trabalhos visam automatizar o processo de defesa, seja através da geração de assinaturas de IDS, geração de alertas pela detecção de anomalias, como também identificação de ataques de último dia através de logs de *honeypots*. O trabalho de Castiglione et al. [20] ainda realiza a configuração de *firewall*, mas baseado em alertas de IDS e no tráfego de dados (*sniffer*). Diferente deste último, o nosso ciclo é inspirado em CA e objetiva alcançar a característica de autoaprendizagem, através da geração automática de novas regras de *firewall* e possível identificação de ataques de último dia. Nosso ciclo ainda pode ser facilmente aplicado em um ambiente de redes com segurança distribuída, bastando seguir os padrões de log.

Em relação ao segundo ciclo autônomo desenvolvido, responsável por manipular dinamicamente *honeypots* virtuais, há trabalhos que visam resolver o problema de *honeypots* comprometidos, porém eles tem uma abordagem passiva em relação a ação a ser tomada nos *hosts* que estão sendo controlados por atacantes para realizar atividades maliciosas a outras máquinas na mesma ou outras redes. Neste sentido, o *Honeywall* [21] foi desenvolvido, um *bootable* CD-ROM capaz de instalar e configurar um centro de controle de *honeynets* junto a um *gateway*. O objetivo dele é proteger a rede de produção de ataques originados a partir dos *honeypots*. Além de utilizar um IDS com módulo reativo (*plugin*) [6], se tornando um IPS, possui regras de *firewall* para controlar o tráfego de saída da *honeynet*.

No mesmo sentido, a ferramenta *Sessionlimit* [88] [44] foi desenvolvida, projetada com o objetivo de conter atividades de intrusos depois que um *honeypot* é comprometido. Ela pode detectar quando há iniciativa de *scan* e DoS, reagindo dinamicamente, inserindo uma nova regra no *firewall* para bloquear o tráfego de saída especificamente da máquina comprometida. Depois de algum tempo a regra é removida. Neste caso o intruso poderá continuar interagindo com o *honeypot*, pois será bloqueado apenas o tráfego de saída originado a partir deste. Contudo, esse nosso segundo ciclo diferencia-se dos demais por ter uma abordagem ativa, realizando a substituição dos *honeypots* virtuais comprometidos.

Esses dois ciclos autonômicos criados tem o intuito de propor soluções para problemas particulares em segurança de redes, mais especificamente na área de metodologias de decepção. Eles são extensões do *framework* desenvolvido. A Tabela 4.1 ilustra uma comparação qualitativa efetuada entre os sistemas de segurança vistos nas seções anteriores com a proposta deste trabalho. Esta comparação leva em consideração se o sistema foi desenvolvido baseado em CA e as propriedades autonômicas oferecidas, ou que há possibilidade de serem oferecidas, como recurso. A nossa proposta é identificada na Tabela 4.1 [91] com a marcação "X".

Propriedades / Tipo	Baseado em CA	Não baseado em CA
<i>Autoconfiguração</i>	[52] [27] [22] [41] X	[94] [73] [72]
<i>Auto-otimização</i>	[52] [22] [41] X	[73] [51] [72]
<i>Autocura</i>	[52] [22] X	[73]
<i>Autoproteção</i>	[52] [27] [22] X	[73]
<i>Autoaprendizagem</i>	[52] [22] [41] X	[94] [73] [51] [72]

Tabela 4.1: Comparação Qualitativa entre Sistemas de Segurança.

Na Tabela 4.1 é possível observar que as propriedades básicas de CA são alcançadas pela proposta deste trabalho. Ou seja, a nossa proposta oferece a possibilidade de alcançar todas as propriedades básicas listadas, através da extensão do *framework*. A capacidade de oferecer como recurso as propriedades autonômicas é uma maneira de qualificar nosso trabalho, podendo levar isso em consideração para caracterizá-lo em relação aos demais trabalhos desenvolvidos anteriormente.

Além de fornecer as propriedades de CA, como visto na Tabela 4.1 e detalhado nos próximos capítulos, este trabalho mostra, através de sua proposta, que é possível obter integração e cooperação entre os sistemas de segurança que foram inicialmente desenvolvidos para trabalharem isoladamente, inteligência através da implantação de estratégias autonômicas que dinamizam o processo de proteção e, por fim, autonomia para alcançar auto-segurança na rede, de maneira que os sistemas consigam se auto-gerenciarem.

4.4 Conclusão

Nesse capítulo inicialmente foram mostrados exemplos de sistemas que se fundamentam na CA, tendo como base os conceitos, arquitetura e propriedades propostos por ela. Em seguida, foram vistos trabalhos que fornecem algumas propriedades da CA de maneira isolada, mesmo não sendo baseados nela. E por fim, a proposta deste trabalho foi detalhada e caracterizada, mostrando o que trouxe de acréscimo para a área de pesquisa e suas diferenças em relação aos demais trabalhos anteriormente desenvolvidos.

5 Segurança Autônômica em Redes de Computadores

Neste capítulo é apresentada a nossa proposta de abordagem autônômica a ser oferecida aos sistemas de segurança de redes. Primeiramente discutiremos as possíveis maneiras de aplicar CA a segurança de redes, tentando mostrar a viabilidade em conseguir atingir segurança autônômica. Em seguida descrevemos, em uma visão de mais alto nível, o nosso mecanismo autônômico, mostrando a ideia da proposta através de sua arquitetura, topologia e funcionamento.

5.1 Aplicabilidade

Os sistemas de proteção das redes atuais estão baseados no paradigma da computação interativa, ou seja, fica a cargo dos administradores humanos decidirem o que fazer e como proteger os sistemas no caso de ocorrências de ataques maliciosos ou erros em cascata imprevistos [19]. Os sistemas que incorporam mais de uma fase para proteção de rede (Seção 3.2), objetivando aumentar ainda mais o nível de segurança, são mais complexos. Isso se deve ao fato de possuírem mais componentes para atenderem os requisitos de cada fase. Essa complexidade necessita de uma constante intervenção humana especializada para a correta utilização do sistema.

Dessa maneira, é interessante a utilização de mecanismos autônômicos para automatizar os processos de gerência. A aplicação dos conceitos da CA à segurança de redes introduz no sistema a capacidade de auto-segurança, através da qual serviços e funções de gerenciamento da segurança são executados sem a necessidade de um gerente humano, a partir da definição de objetivos e parâmetros iniciais fornecidos por seus administradores [84].

Para tentar propor soluções aos problemas descritos anteriormente (Seção 3.3), é possível destacar também que a arquitetura de sistemas autônômicos pode ser aplicada ao desenvolvimento de software voltado para defesa e segurança. O modelo

MAPE-K (Seção 2.4) oferece uma visão conceitual de como sistemas autônomicos podem ser desenvolvidos para atender necessidades de segurança.

Sensores podem ser qualquer programa que verifique ocorrências de tráfego malicioso, independente de qual fase de proteção seja, coletando informações da rede para serem enviadas aos monitores. Exemplo de dados coletados podem ser, por exemplo, o tráfego destinado a *honeypots*, alertas de IDS, logs de *firewall*, etc. Os monitores irão receber essas informações e tratá-las a fim de extrair o que vem a ser relevante, por exemplo, o endereço IP de origem do intruso, o protocolo utilizado pelo serviço no qual foi realizado o ataque, horário/data da intrusão, etc.

Os monitores enviam as informações necessárias para os componentes de análise e planejamento, que as utilizam para seu processamento. Como se sabe, geralmente as fases de análise e planejamento são implementadas em um único componente. O processamento realizado por esse componente varia de acordo com a estratégia autônoma adotada pelo administrador que inseriu os objetivos para o sistema. Um exemplo de processamento utilizando *ECA rules* pode ser visto na Figura 5.1. Neste exemplo, na ocorrência de um alerta de IDS (*IDSAlert*), se o IP de origem que gerou o alerta não estiver na lista negra do *firewall* (*BlackList !Contains SrcIpAddr*), será adicionado (*Add SrcIpAddr in BlackList*).

```
on IDSAlert if BlackList !Contains SrcIpAddr do Add SrcIpAddr in BlackList
```

Figura 5.1: Exemplo de reconfiguração.

A base de conhecimento pode ser utilizada pelo componente de análise e planejamento na adoção de estratégias para, por exemplo, não realizar ações anteriormente já feitas. Na Figura 5.1 não serão adicionados endereços IP na lista negra que já estiverem contidos. O componente de análise e planejamento tem como saída um plano de ação a ser executado pelo componente executor, que na Figura 5.1 é a adição do endereço IP de origem que gerou o alerta na lista negra.

O executor aplica as ações no elemento gerenciado através dos atuadores. Atuadores são responsáveis por fazerem alterações de configuração no elemento gerenciado, ou seja, em alguma aplicação de segurança da rede. O objetivo em realizar as mudanças nas configurações é aumentar o nível de segurança. No exemplo da

Figura 5.1 o atuador interage diretamente com a aplicação responsável pela lista negra, o *firewall*.

Além da arquitetura de sistemas oferecida pela CA, visando o estado em que se encontram os atuais sistemas de segurança e também o crescimento e evolução dos ataques às redes de computadores, é possível relacionar a necessidade que este tipo de sistema tem pelas propriedades dos sistemas autônômicos, destacando as propriedades de autoproteção (*self-protecting*), autocura (*self-healing*), autoconfiguração (*self-configuring*) e autoaprendizagem (*self-learning*) [74], como visto a seguir:

- **Autoproteção:** Refere-se à propriedade do sistema de defender-se de ataques acidentais ou maliciosos. Para tanto, o sistema deve ter conhecimento sobre potenciais ameaças, bem como prover mecanismos para tratá-las [23]. Para atingir essa propriedade o sistema deve ter habilidade para antecipar, detectar, identificar e proteger o elemento gerenciado contra ameaças;
- **Autocura:** Responsável por identificar e corrigir erros ou falhas. No contexto da segurança de redes, o sistema autônômico deve ser capaz de detectar, diagnosticar e consertar problemas resultantes de ataques a ativos de produção da rede. O sistema deve possuir um componente de diagnóstico que analisa informações que mostram a ocorrência de falhas ou danos causados por um ataque à rede, para isto deve utilizar conhecimento sobre a configuração dos recursos da rede. Então posteriormente deve buscar uma solução a ser tomada, aplicá-la e depois testar se foi satisfatória. O processo de cura deve ser realizado com máxima transparência para usuários legítimos da rede;
- **Autoconfiguração:** Nesta propriedade é fornecida a capacidade para os sistemas se configurarem e reconfigurarem automaticamente de acordo com políticas de negócio fornecidas por seus administradores, os quais definem o que deve ser feito e não como fazer [23]. A fim de automatizar o gerenciamento de configuração, um sistema de segurança deve possuir capacidade de reconfiguração dinâmica com o mínimo de intervenção humana;
- **Autoaprendizagem:** Propriedade que fornece capacidade de aprendizado a partir de dados sensoreados e também através de experiências e resultados obtidos em ações anteriores. Propriedade fundamental para sistemas de

segurança, visto que ela oferece a capacidade para o sistema aprender a se defender de ataques antes desconhecidos, ou, pelo menos, reconhecer tráfego malicioso na rede para posterior defesa.

Mesmo diante de várias qualidades oferecidas pela CA, aplicá-la às redes de computadores e sua segurança não é um processo trivial, há desafios a serem enfrentados para alcançar o auto-gerenciamento. Segundo Agoulmine et al., em [14], o desafio é simplificar as tarefas de gerência para o administrador, automatizando o processo de decisão. Questões de segurança é outro desafio para o desenvolvimento de sistemas autônômicos, visto que o emprego de autoaprendizagem e auto-evolução poderá ocasionar a perda do domínio humano sob a gerência das decisões tomadas pelo próprio sistema, havendo a possibilidade de desvio dos objetivos iniciais inseridos. Com isso, no processo de desenvolvimento de software autônômico a fase de validação deve ser aplicada de maneira rigorosa.

5.2 Uma proposta de mecanismo autônômico para Segurança de Redes

O nosso trabalho é inspirado na ideia de CA e sua utilização para resolver problemas em segurança de redes. Diante de todo conteúdo abordado anteriormente, acreditamos na aplicação dos conceitos e princípios da CA para se obter ambientes de redes seguros. Então propomos primeiramente um *framework* autônômico com a finalidade de ser reutilizado em ambientes de redes por sistemas de segurança. Em seguida utilizamos o próprio *framework* para propor dois ciclos autônômicos que utilizam como base *honeypots*. O conjunto composto pelo *framework* e pelos dois ciclos autônômicos denominamos de Mecanismo Autônômico para Segurança de Redes baseado em Decepção, nome inspirado na ideia de Metodologias de Decepção (Seção 3.2.2), e o chamamos de *AutonomicSec* (*Autonomic Security* - Segurança Autônômica). Nas próximas seções apresentamos detalhes da arquitetura, topologia e funcionamento da proposta desse mecanismo.

5.2.1 *Framework* Autônomo

A nossa proposta de *framework* é organizada seguindo o modelo MAPE-K para fornecer a possibilidade de alcançar integração, cooperação, inteligência e, principalmente, autonomia aos sistemas de segurança de redes. Neste modelo gerentes autônomos realizam as atividades de sensoriamento do ambiente de execução, análise de contexto, planejamento e execução de ações de reconfiguração dinâmica. A arquitetura do *framework* pode ser vista na Figura 5.2.

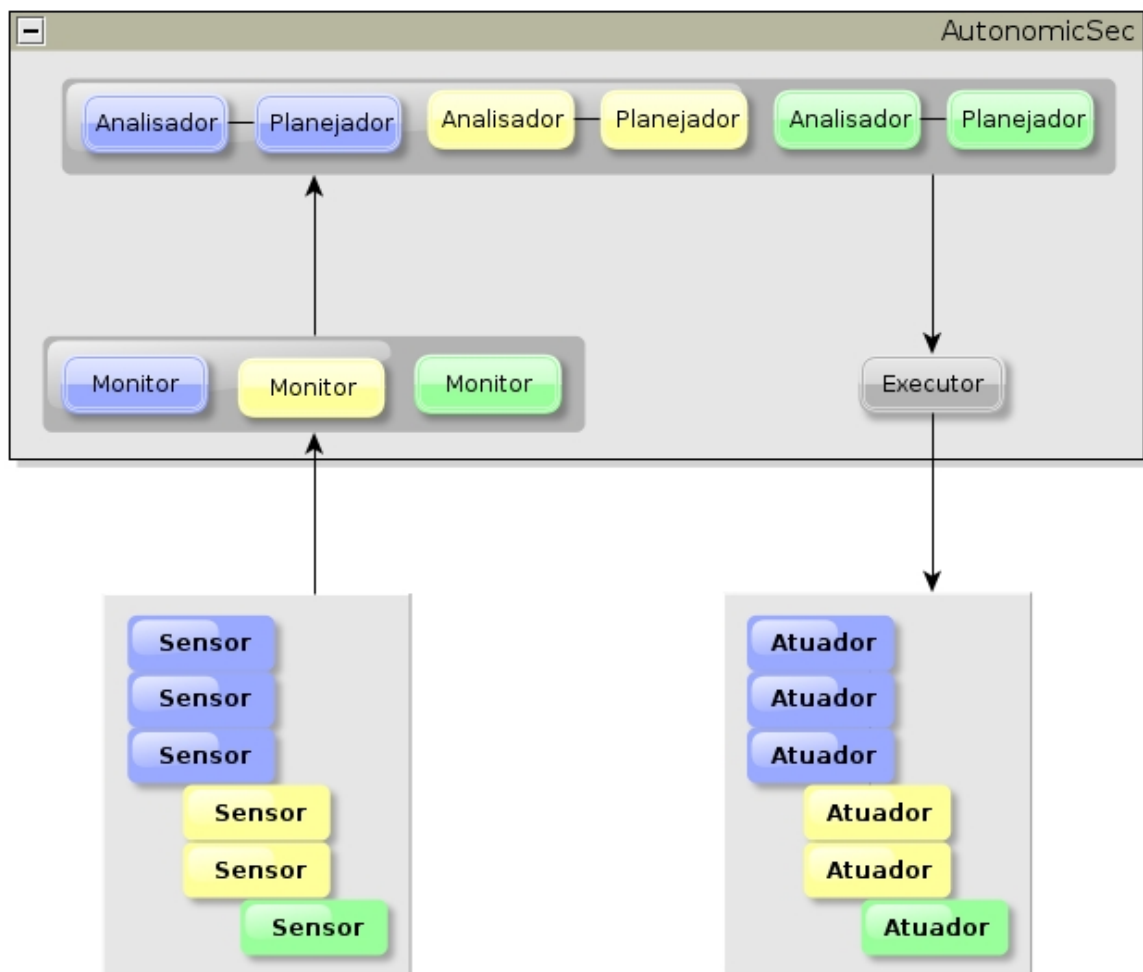


Figura 5.2: Arquitetura do *Framework*.

Como visto na arquitetura do *framework*, sensores são componentes de software que podem coletar dados de qualquer natureza, sendo o tipo desses dados que determina o próprio tipo do sensor. O número de sensores é extensível, podendo haver crescimento na quantidade de dados sensoreados. Cada sensor necessita passar para o *AutonomicSec*, além das suas informações coletadas, o tipo (identificador) do monitor no qual é responsável por tratar o dado coletado. Portanto, a comunicação

remota entre sensores e *AutonomicSec* transfere dois campos: o próprio dado coletado e o identificador do monitor que tem a capacidade de tratar esses dados. Com isso, é possível ter vários tipos de sensores ligados a vários tipos de monitores. Porém, cada sensor necessariamente deve possuir uma ligação com pelo menos um monitor, caso contrário os dados sensoreados serão descartados pelo *AutonomicSec*.

Essa maneira nos quais sensores e monitores se comunicam fornece possibilidade de crescimento e organização para o sistema. Por exemplo, tendo como base a Figura 5.2, uma rede possui três sistemas de segurança onde é possível coletar dados para, a partir desses, realizar alguma tomada de decisão, são eles: três sensores A, dois sensores B e um sensor C. Cada sensor envia para o *AutonomicSec* o dado coletado e o seu tipo. Então o *AutonomicSec* fica responsável por passar os dados coletados pelos sensores do tipo A para o monitor A, dos sensores do tipo B para o monitor B e do sensor C para o monitor C. Em cada monitor é criada um fila de requisições dos sensores contendo os dados que devem ser filtrados. Caso não haja nenhuma informação a ser tratada por um monitor, este componente (processo) ficará em modo de espera (*wait*).

Internamente ao *AutonomicSec*, logo que ele é inicializado, é feita uma ligação entre componentes, através da passagem da referência (nome) de suas instâncias. Chamamos isso de gerenciamento de registros de componentes. O monitor inicialmente é registrado no servidor *AutonomicSec*, que esperará por requisições *sockets* de sensores, os quais enviarão dados coletados através dessas conexões. O analisador e planejador são implementados como um único componente, assim ele deve ser registrado em pelo menos um monitor. Por fim, o executor, único no sistema, é registrado em todos os componentes de análise e planejamento. Dessa forma o monitor sabe que deve encaminhar os dados relevantes para um determinado analisador/planejador, que por sua vez irá passar um plano de ações de reconfiguração para serem tomados pelo executor, como pode ser visto na Figura 5.3.

O plano de ações resultante do processamento realizado pelo componente de análise e planejamento é baseado no que chamamos de estratégia autônoma. Através desta última são geradas as decisões sobre quais modificações serão realizadas no elemento gerenciado. Estas modificações correspondem às operações de reconfiguração que, de acordo com o mecanismo de tomada de decisão, devem ser

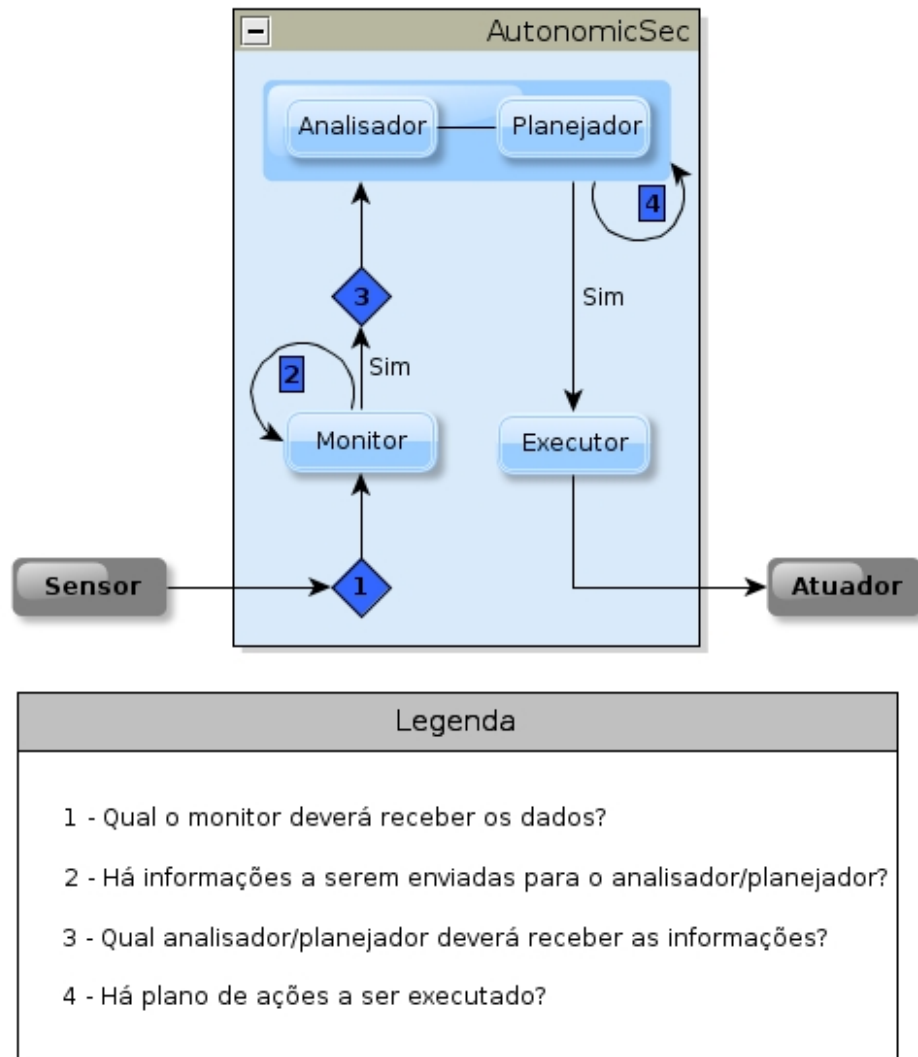


Figura 5.3: Comunicação entre componentes do *Framework*.

executadas. A estratégia autônoma adotada pela fase de análise e planejamento deve levar em consideração os objetivos do ciclo a qual ela faz parte.

O executor é único para todo o sistema e processa as ações em série, sendo estas as características fundamentais para este componente, pois as ações de reconfiguração no elemento gerenciado podem ser conflitantes. Igualmente aos monitores, no componente de análise e planejamento é criada uma fila, mas esta é contida de dados filtrados e considerados relevantes. E no executor também é criada uma fila, contendo plano de ações a serem executadas. Logo, quando não há nenhum dado a ser processado pela fase de análise e planejamento ou plano de ações a ser tomado pelo executor, estes entram em modo de espera.

O executor tem a função de simplesmente executar os planos de ações enviados pelos analisadores/planejadores. Ele aplica ações no elemento gerenciado através dos atuadores. Estes, por sua vez, interagem diretamente com o elemento gerenciado. Os atuadores podem modificar, por exemplo, a configuração de algum sistema de segurança da rede, com o objetivo de aumentar o nível de dificuldade para a concretização de ataques. Como visto na Figura 5.2, podem existir vários tipos de atuadores e em quantidade escalável, considerando que o elemento gerenciado é a rede e não sistemas isolados. Diferentemente da ligação que existe entre sensores e monitores, os planos de ações podem ser executados através de qualquer atuador, independente do seu tipo ou da sua quantidade, dependendo apenas da tomada de decisão realizada pelo componente de análise e planejamento.

A abordagem que utilizamos para a nossa proposta de *framework* foi tentar seguir fielmente o modelo MAPE-K, considerando todos seus componentes e a base de conhecimento (*knowledge*). Esta última representada pela estratégia autônoma. Consideramos e acreditamos que nosso *framework* possa ser utilizado em diversos ambientes de rede para a integração e cooperação entre sistemas, mais especificamente entre sistemas de segurança de redes.

Para mostrar a viabilidade de utilização do *framework*, propomos dois ciclos autônicos focados em segurança de redes, que estendem todos seus componentes. O processo de criação de ciclos autônicos através do reuso do *framework* é descrito no próximo capítulo. Os ciclos propostos são detalhados a seguir.

5.2.2 Primeiro Ciclo Autônomo

Com a utilização do *framework*, propomos o primeiro ciclo autônomo. Este ciclo tem a funcionalidade de gerar regras de *firewall* baseadas em logs de *honeypots*. Como visto na Figura 5.4, os componentes sensores são representados por *honeypots*, ou seja, os dados sensoreados são logs a nível de rede gerados por *honeypots* de baixa interatividade. Esses logs registram interações com o *honeypot* na camada de rede. Então qualquer interação com um *honeypot* gera log e esse é enviado para o componente monitor, no *AutonomicSec*.

O monitor recebe os dados sensoreados e realiza uma filtragem do que é relevante para ser enviado à fase de análise e planejamento. Os dados recebidos

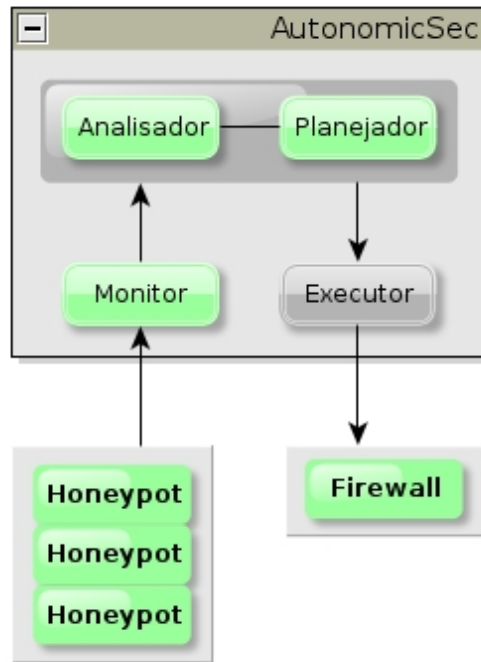


Figura 5.4: Arquitetura do Primeiro Ciclo Autônomo.

são: endereço IP de origem (possível intruso), IP de destino (*honeypot* que recebeu interação), porta de origem, porta de destino (informa o serviço), *timestamp* (horário e data) e protocolo. Este último podendo ser: *Transmission Control Protocol* (TCP), *User Datagram Protocol* (UDP), *Internet Control Message Protocol* (ICMP) ou *Internet Group Management Protocol* (IGMP). Para esse ciclo autônomo são descartados: *timestamp*, porta de origem e interações com o protocolo IGMP, pois não são informações utilizadas pela estratégia autônoma. Dessa maneira, são enviadas apenas informações úteis.

Nós chamamos a tomada de decisão realizada no componente de análise e planejamento de **estratégia autônoma baseada em intenções**. As regras de *firewall* geradas por ela são de acordo com a intenção do intruso ao interagir com um *honeypot*. Caso a interação com o *honeypot* seja a um determinado serviço, o atacante terá acesso bloqueado a todos os serviços equivalentes da rede de produção. Como visto na Figura 5.5, os *honeypots* emulam dispositivos e serviços equivalentes aos servidores de produção. Em suma, é criada uma *honeynet*, sendo esta uma imagem da rede de produção. Na ocorrência de uma interação com um serviço A emulado por *honeypot*, o usuário malicioso terá acesso rejeitado a todos serviços do tipo A na rede de produção.

Esse processo é realizado de maneira transparente para qualquer usuário, inclusive o próprio intruso, tendo como base sua intenção.

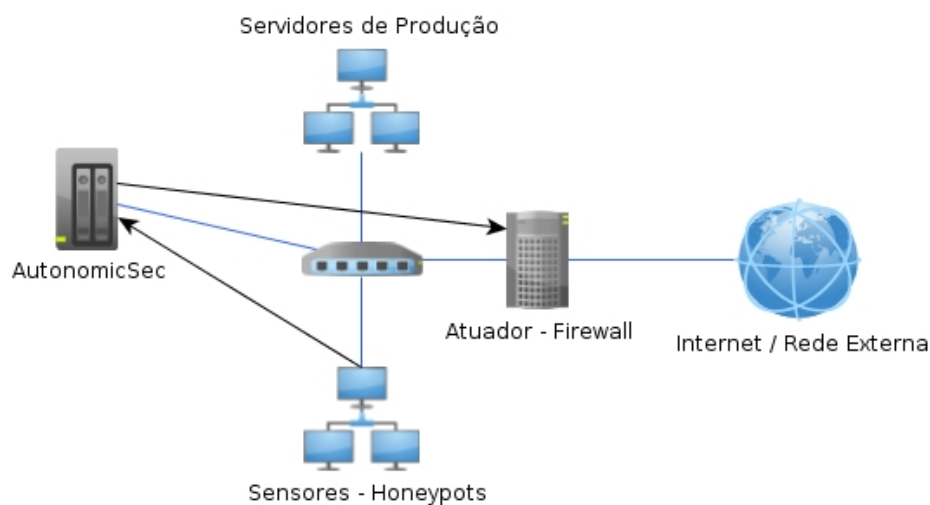


Figura 5.5: Topologia do Primeiro Ciclo Autônomo.

A exemplo de uso da estratégia autônoma baseada em intenções, um usuário mal intencionado está objetivando realizar um ataque ao servidor Web da rede. Ele interage primeiramente com a *honeypot* que emula o respectivo serviço de porta 80, utilizada normalmente por servidores Web. Com isso, será gerada uma regra a ser aplicada pelo executor, através do atuador, na tabela de encaminhamento em uma aplicação de *firewall*. Esta regra modifica as permissões de acesso do endereço do intruso, de maneira que ele seja rejeitado em qualquer tentativa de acesso (ou ataque) a todos serviços Web de porta 80 que possam existir na rede de produção. Portanto, nossa estratégia é aplicar regras para controle de tráfego entre redes distintas, associadas a modificação das permissões de acesso de usuários externos a serviços da rede interna, como pode ser visto na Figura 5.6. No caso de interação com um *honeypot* com o uso do protocolo ICMP, utilizado pelo *ping*, o intruso tem acesso bloqueado nesse protocolo a todos servidores de produção da rede, não levando-se em consideração a porta. Nós chamamos o conjunto de endereços bloqueados dessa forma de **lista cinza**.

Uma outra tomada de decisão feita por nossa estratégia autônoma baseada em intenções é a verificação da ocorrência de endereços, a fim de inspecionar a existência de uma quantidade elevada de regras geradas por uma determinada origem. Se houver ocorrências acima de um valor especificado na configuração deste ciclo, o

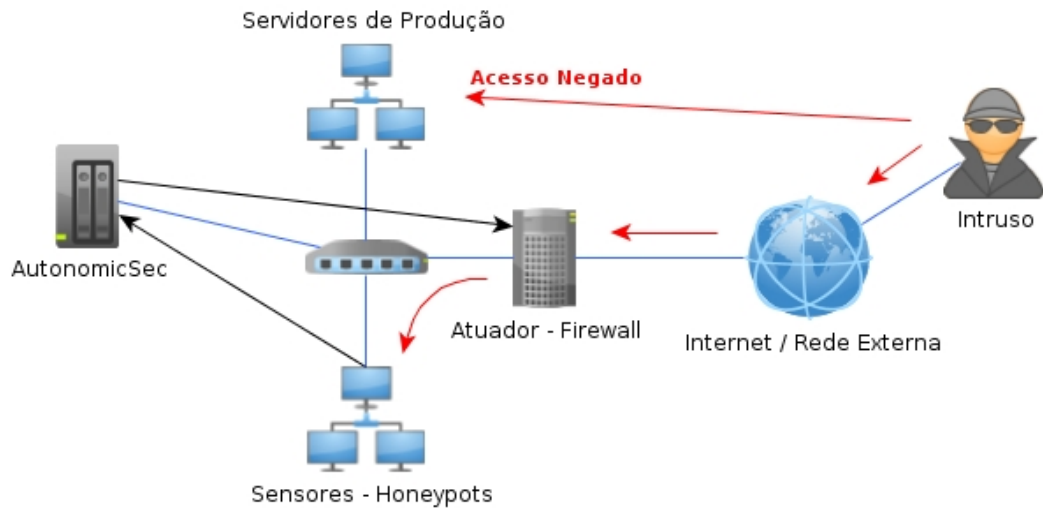


Figura 5.6: Lista Cinza.

endereço não terá permissão negada apenas para determinados serviços contidos em regras geradas por ele, mas a toda rede. Nesse caso consideramos que sua intenção não é apenas atacar um serviço específico, mas sim qualquer vulnerabilidade que haja na rede. Como visto na Figura 5.7 por exemplo, o intruso tem acesso negado a toda a rede se interagir com três serviços distintos de *honeypots*, passando a integrar o que chamamos de **lista negra**. Dessa forma, é feita uma otimização nas regras do *firewall*, excluindo todas regras de bloqueio a serviços específicos geradas anteriormente e criando apenas uma de bloqueio total a rede.

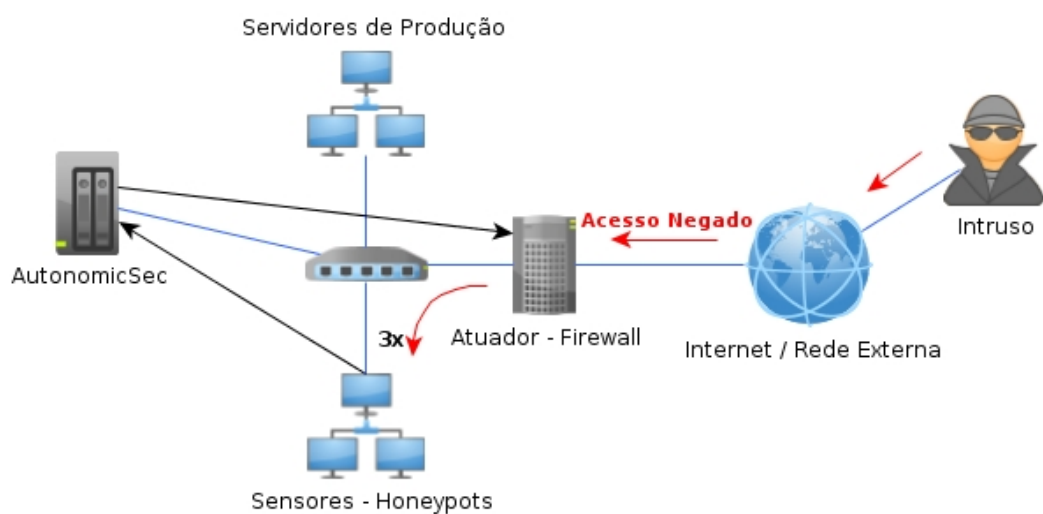


Figura 5.7: Lista Negra.

A estratégia autônoma necessita de algumas informações para funcionar corretamente, sendo uma delas a de identificação dos servidores de produção da rede, na qual deve conter o endereço, protocolo (TCP, UDP ou ICMP) e porta utilizada pelo serviço. Uma outra informação que a estratégia deve conhecer são os endereços considerados sempre legítimos na rede, nos quais fazem parte do que chamamos de **lista branca**. Nela são informados endereços que não geram regras de *firewall*, pois são considerados confiáveis, mesmo que interajam com um *honeypot* e gerem log no componente sensor.

Ao final deste ciclo obtemos as seguintes propriedades autônomas: autoconfiguração, pois este ciclo do *AutonomicSec* reconfigura dinamicamente as regras de controle de tráfego da aplicação de *firewall*; autoaprendizagem para o próprio *firewall*, porque as regras são implementadas nele a fim de que possa adquirir conhecimento sobre novos usuários mal intencionados e assim bloqueá-los, de acordo com sua intenção de ataque; auto-otimização para as próprias regras, visto que no caso da inserção de um endereço na lista negra, há uma redução na sua quantidade, retirando as que rejeitam para serviços específicos e adicionando apenas a que bloqueia para toda rede interna; por fim, autoproteção, sendo esta a propriedade implícita a ser atingida pela rede e objetivo deste primeiro ciclo autônomo.

5.2.3 Segundo Ciclo Autônomo

O segundo ciclo autônomo também é proposto utilizando o *framework*, como uma extensão. Ele é responsável por manipular dinamicamente *honeypots* virtuais comprometidos com base em logs de *firewall*. Como pode ser visto na Figura 5.8, o sensor é representado pelo *firewall* e o atuador pelo hipervisor dos *honeypots* virtuais. O uso de *honeypots* combinado com a tecnologia de virtualização em segurança de redes vem sendo explorado a fim de obter uma melhor utilização dos recursos oferecidos [53]. A implantação de uma *honeynet* implica em alocar vários computadores, com sistemas operacionais e serviços distintos, o que pode significar um alto custo de implantação e operação. Nesse contexto, máquinas virtuais podem ser usadas para construir *honeynets* virtuais. Para esta situação a virtualização traz vantagens, como o menor custo de implantação e gerência.

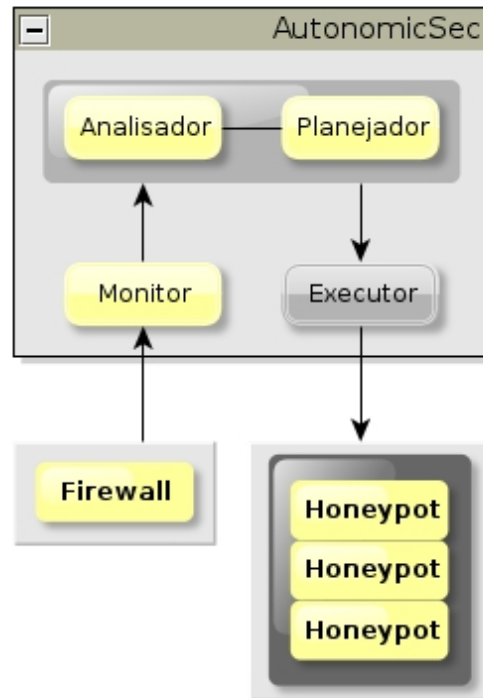


Figura 5.8: Arquitetura do Segundo Ciclo Autônomo.

Um *honeypot* é considerado comprometido quando o intruso que interage com ele descobre que está sendo enganado, ganha domínio e passa a utilizá-lo como intermediador para a realização de outros ataques [86] [59] [17]. Normalmente *honeypots* comprometidos, quando dominados (invadidos), são usados como nós integrantes de uma *botnet* e utilizados para prejudicar outros sistemas. Dessa forma, o problema de *honeypots* comprometidos é um risco que diminui o potencial da ideia de metodologias de decepção para a área de segurança, no qual merece atenção e deve ser tratado.

Neste ciclo, o sensoriamento são em logs gerados por uma aplicação de *firewall* para verificar a ocorrência de tráfego de saída originados a partir de *honeypots*. O *firewall* deverá está configurado para registrar no log tentativas de início de conexão que utilizam os protocolos TCP, UDP e ICMP que excedam uma determinada quantidade. Ou seja, o *honeypot* só poderá receber conexões UDP e ICMP, como também responder conexões TCP que tenham sido iniciadas a partir de um usuário externo. A configuração inserida no *firewall* é responsável por determinar a quantidade limite de conexões que podem ser iniciadas a partir do *honeypot* em um intervalo de tempo, por exemplo, dez conexões por hora. Dessa maneira é possível identificar uma máquina que está sendo utilizada para intermediar a realização de ataques,

pois um *honeypot* não deve ser instalado e mantido de forma que inicie interações (conexões). Portanto, se um *honeypot* iniciar conexões acima de um limite especificado, ele é considerado comprometido. Esse mecanismo de identificação de *honeypots* comprometidos já é implementado no *Honeywall* e *sessionlimit*, então tomamos como base na geração dos logs para os sensores deste ciclo.

Os sensores enviam para o *AutonomicSec* os logs gerados pelo *firewall*, nas ocorrências explicadas acima, que as repassam ao monitor responsável. Este, por sua vez, extrai o endereço do nó da rede que gerou a ocorrência e o envia para o componente de análise e planejamento. Portanto a filtragem realizada pela fase de monitoramento neste ciclo é simples, havendo a necessidade de extrair apenas o campo de endereçamento. Dessa maneira é possível identificar um *honeypot* comprometido, se o log tiver sido gerado pelo endereço de um e isso é feito pela estratégia autônoma deste ciclo.

Então na próxima etapa, a de análise e planejamento, a estratégia autônoma verifica se o endereço recebido pertence a alguma máquina virtual da *honeynet*. Caso isso ocorra, a ação a ser executada através do atuador, que neste ciclo é representado pelo hipervisor dos *honeypots* virtuais, como visto na Figura 5.9, é a substituição dos *honeypots* comprometidos. Essa substituição é dinâmica e representa o desligamento da máquina virtual e inicialização de outra idêntica com as mesmas características, porém, sem estar sob domínio do intruso. Consideramos que qualquer código malicioso instalado inicialmente pelo usuário mal intencionado em um *honeypot* para ser utilizado como ponte para a realização de outros ataques não terá efeito e, assim, não será mais dominado, visto que a máquina virtual de decepção será desativada junto com qualquer atividade maliciosa realizada anteriormente. Portanto, o plano de ações neste ciclo é a realocação dinâmica de *honeypots*, substituindo os comprometidos por suas réplicas não infectadas.

Para o correto funcionamento deste ciclo, informações da *honeynet* necessitam ser conhecidas, a saber: endereço da máquina do hipervisor responsável pelo gerenciamento da virtualização (pode haver mais de um), endereços dos *honeypots* para a identificação dos mesmos (quando gerarem log de tráfego de saída no *firewall*) e localização das imagens de cada máquina virtual que representa a réplica não infectada por códigos maliciosos. Esta última é criada inicialmente antes do *honeypot* ser colocado em atividade, não havendo a possibilidade de está comprometida.

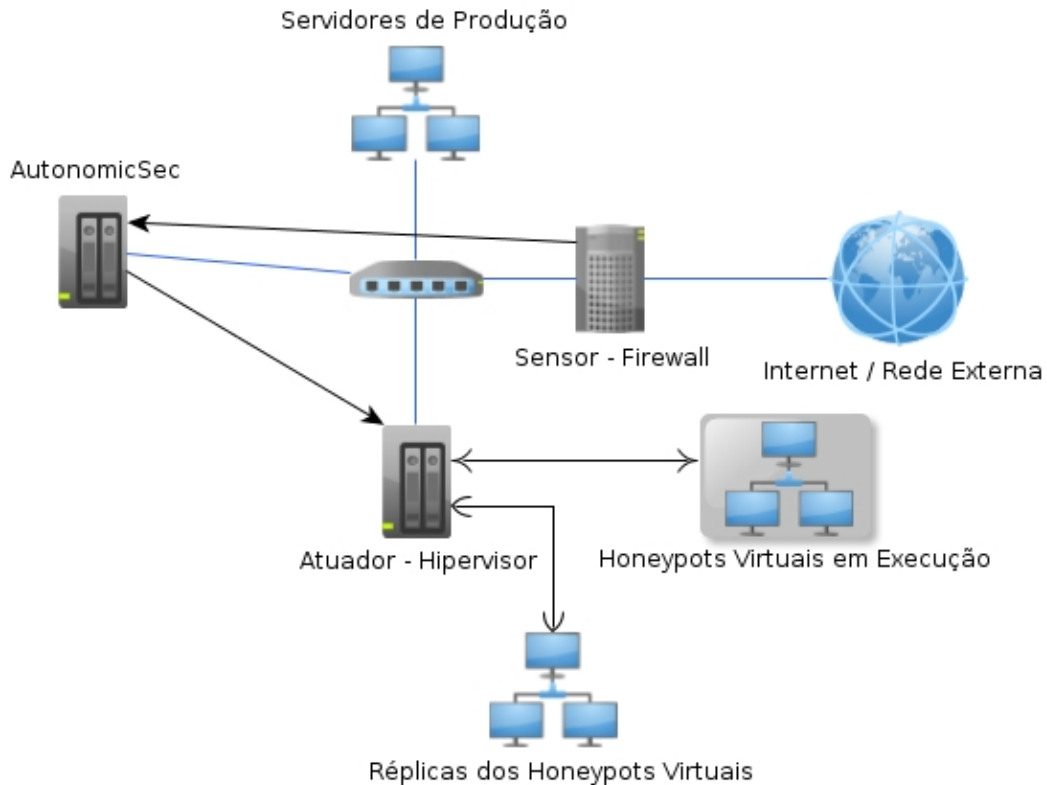


Figura 5.9: Topologia do Segundo Ciclo Autônomo.

Ao final deste ciclo obtemos as seguintes propriedades autônomas: autoconfiguração, pois este ciclo do *AutonomicSec* reconfigura dinamicamente a *honeynet*, através da manipulação dos *honeypots* virtuais; autoproteção para a rede como um todo, partindo do princípio que metodologias de decepção não serão utilizadas para a realização de ataques à rede externa e também interna; por último, autocura para os próprios *honeypots*, desabilitando o infectado e inserindo um que não esteja, sendo este o processo de cura. Contudo, consideramos que um mecanismo de decepção deve ser, na verdade, utilizado para a proteção de redes na fase de detecção de intrusão e não um meio para a realização de ataques, sendo este segundo ciclo focado em conseguir uma solução que garanta isso.

5.2.4 Discussão

Para introduzir um novo ciclo é necessário estender os três principais componentes do *AutonomicSec*: monitor, analisador/planejador e o plano de ação, que é o código a ser processado pelo executor. Como também devem ser implementados

os componentes sensor e atuador. O sensor deve ser criado como um cliente *socket* coletando dados e os enviando ao monitor. O atuador, também externo ao *framework*, poderá ser qualquer aplicação responsável pela reconfiguração do elemento gerenciado. O monitor deve ser implementado de maneira a filtrar o que é relevante para a tomada de decisão, passando dados úteis para o componente de análise e planejamento. Este último, por sua vez, possuirá o algoritmo da estratégia autônoma. E o plano de ações, terá o código processado pelo executor.

Ressaltamos que nosso *framework* é criado com o objetivo de ser aplicado em redes, para cooperação entre sistemas de segurança, uma vez que foi pensado com intuito de resolver problemas nesses ambientes. Isso não significa que ele não pode ser estendido ou adaptação para problemas em outras áreas. Com o reuso dele o primeiro ciclo autônomo foi implementado, tendo como intenção a melhor utilização dos logs gerados por *honeypots*. Ele foi inspirado em CA e objetiva alcançar a característica de autoaprendizagem, utilizando regras ECA, através da geração automática de novas regras de *firewall* e possível identificação de ataques de último dia. O segundo ciclo autônomo foi desenvolvido, também com o reuso do *framework*, para contornar o problema de *honeypots* comprometidos, através da manipulação dinâmica de máquinas virtuais. Ele possui uma abordagem ativa para o problema em questão, utilizando-se para isso a substituição dos *honeypots* virtuais comprometidos.

Reiteramos ainda que esses dois ciclos autônomos propõem soluções para problemas particulares na área de metodologias de decepção e que as propriedades básicas de CA são alcançadas por ele: autoconfiguração, autocura, autoaprendizagem, auto-otimização e autoproteção. Além disto, a proposta deste trabalho (*framework* e ciclos autônomos) mostra que é possível obter: integração e cooperação entre sistemas de segurança, que foram inicialmente desenvolvidos para trabalharem isoladamente; inteligência, através da implantação de estratégias autônomas que dinamizam o processo de proteção; e autonomia, para alcançar auto-segurança na rede de maneira que os sistemas consigam se auto-gerenciarem.

5.3 Conclusão

Este capítulo apresentou o *AutonomicSec*, o nosso mecanismo autônomo para segurança de redes baseado em decepção. Foi visto a aplicabilidade da CA para a área de segurança de redes, de maneira que foi possível verificar a viabilidade disto. Em seguida foi detalhada toda a ideia de nossa proposta, através da explicação da arquitetura, topologia e seu funcionamento. Para isso, dividimos o mecanismo nos três tópicos: *framework* autônomo, primeiro ciclo autônomo e segundo ciclo autônomo, sendo que os dois ciclos autônomos são uma extensão do *framework*. Além disso, foi possível observar o principal contexto em que o *AutonomicSec* é aplicável. E ressaltamos que o *framework* permite, através de seu reuso, ser estendido para criar outros ciclos autônomos.

6 Avaliação do Mecanismo Autônomo

Este capítulo descreve a avaliação de nossa proposta. Para isso, desenvolvemos um protótipo do mecanismo autônomo, através da implementação do *framework* e sua extensão para os dois ciclos autônicos. Posteriormente, realizamos testes nos ciclos em cenário comum de uso e, em seguida, detalhamos os resultados obtidos em cada componente correspondente a uma fase do ciclo MAPE-K. Ao explicar os resultados alcançados, mostramos o que o mecanismo autônomo agrega para a segurança de redes.

6.1 Implementação do *Framework* Autônomo

A fim de tornar a ideia do *framework* autônomo utilizável, para ser estendido em aplicações de segurança, o implementamos utilizando tecnologia Java [7] e o Ambiente Integrado de Desenvolvimento - *Integrated Development Environment* (IDE) - Eclipse [2]. A Figura 6.1 mostra o diagrama de componentes, na qual é possível observar como os componentes são distribuídos (sistema descentralizado).

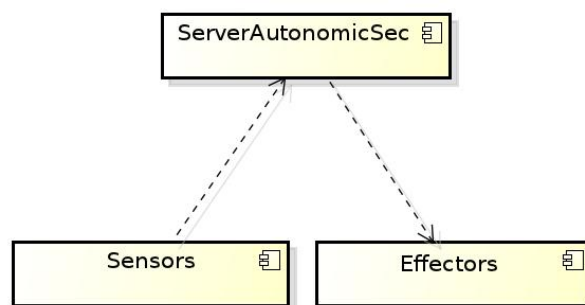


Figura 6.1: Diagrama de componentes do *Framework*.

Os componentes de sensoriamento (*sensors*) devem ser implementados junto a sistemas de segurança, para serem utilizados na coleta de dados em uma extensão. Da mesma maneira que os componentes atuadores (*effectors*) precisam ser implementados junto a outros sistemas de segurança, sendo que estes, no entanto, sofrerão reconfigurações. Como explicado anteriormente na Seção 5.2.1, ressaltamos que ao implementar cada sensor é necessário passar para o *AutonomicSec* dois campos:

o próprio dado coletado e o identificador do monitor que tem a capacidade de tratar esses dados. O componente *ServerAutonomicSec* é implementado em um servidor, como o próprio nome sugere, de maneira a ficar responsável pela execução do restante do ciclo.

A nossa implementação preza por simplicidade, com o objetivo de facilitar a utilização de CA, através do uso do modelo MAPE-K. Com isso, o *framework* impõe uma arquitetura a ser seguida, de maneira que em todas extensões, ou ciclos, cada componente execute funcionalidades que sejam de sua responsabilidade. Com isso, o diagrama de classe pode ser visto na Figura 6.2.

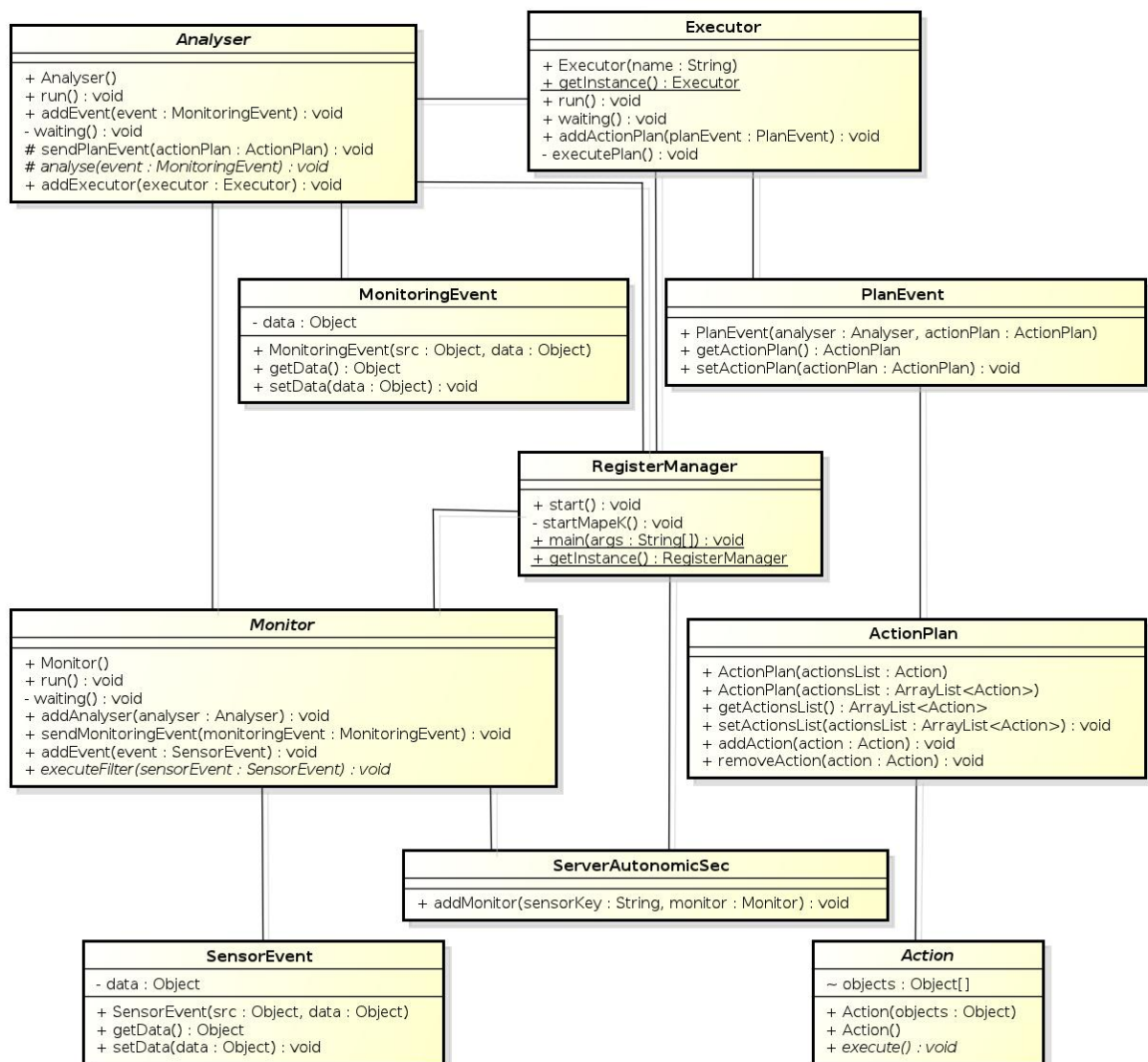


Figura 6.2: Diagrama de classe do *Framework*.

A classe *RegisterManager* inicializa o *framework*, juntamente com todos os processos, representados em Java por *threads*, que a compõem, através do

método `startMapeK()`, são eles: `ServerAutonomicSec`, `Monitor`, `Analyser` e `Executor`. A funcionalidade desta classe é a criação do registro, ou associação, entre componentes, onde monitores são registrados no servidor, analisadores/planejadores nos monitores, que por sua vez são associados ao executor, de maneira que cada componente saiba para onde enviar o resultado de seu processamento, criando assim a ideia de ciclo. Além disso, esta classe implementa o padrão de projeto *Singleton* [32], através do método `getInstance()`, garantindo que apenas um objeto possa ser instanciado a partir dessa classe.

A classe `ServerAutonomicSec` é responsável pela criação do servidor *socket*, onde sensores irão conectar para o envio de dados sensoreados a monitores. Dessa maneira, através do método `addMonitor(String sensorKey, Monitor monitor)`, devem ser adicionados todos os monitores da aplicação. Além disso, essa classe implementa a estrutura de decisão para a escolha do monitor correto que irá tratar os dados recebidos dos sensores. Essa tomada de decisão é feita de acordo com o parâmetro `sensorKey`, utilizado ao instanciar monitores e adicioná-los no servidor, como também passado por sensores para serem associados a esses últimos.

A filtragem nos dados recebidos por sensores é de responsabilidade da classe `Monitor`, realizada pelo método `executeFilter(SensorEvent sensorEvent)`. Ela é instanciada como processo (herda de `Thread`), onde eventos sensorizados são depositados em uma fila por sensores, com o uso do método `addEvent(SensorEvent event)`. Quando esta fila está vazia, o processo entra em estado de espera (`waiting()`), sendo notificado para entrar em execução quando é inserido algum evento nessa fila. Após os filtros serem realizados, as informações úteis serão enviadas para todos analisadores registrados em um monitor. O envio é feito pelo método `sendMonitoringEvent(MonitoringEvent monitoringEvent)` e o registro pelo `addAnalyser(Analyser analyser)`.

Na classe `Analyser`, instanciada também como processo, o método `analyse(MonitoringEvent event)` é o responsável por executar a estratégia autônômica utilizada pelo ciclo, o que normalmente necessita acessar dados em arquivos de configuração e/ou uma base de conhecimento (*knowledge*) para tomada de decisões. O executor é conhecido por esta classe através do método `addExecutor(Executor executor)`, no qual determina o fluxo do ciclo. A manipulação da fila contendo dados filtrados pelo monitor é realizada por dois

métodos: `addEvent(MonitoringEvent event)`, para a inserção, e pelo próprio `analyse(MonitoringEvent event)`, para remoção, quando for processar a estratégia autônômica. Caso a fila esteja vazia, a *thread* irá esperar, com a chamada do método `waiting()`.

Ainda na classe `Analyser`, o método `sendPlanEvent(ActionPlan actionPlan)` envia eventos para o executor. Um evento é composto por uma ou mais ações (`Action`), resultado do processamento realizado pela estratégia autônômica, e que formam um plano de ações. Esse último é manipulado através da classe `ActionPlan`, na qual o analisador pode adicionar ou remover ações em uma lista.

Por fim, a classe `Executor`, que também herda de `Thread`, realiza a execução de todas as ações enviadas a ela. Em um ambiente de redes, o componente executor normalmente aplica ações de reconfiguração nos sistemas através da passagem de parâmetros para scripts. Esses scripts representam os atuadores e são usados para modificar, inserir ou remover configurações em sistemas de segurança de redes. No caso dos atuadores do *AutonomicSec*, os parâmetros são passados a eles remotamente, através do protocolo *Secure Shell* (SSH) [9] e, como os sistemas de segurança executam em sistema operacional Linux, os scripts são, na verdade, escritos em *Shell Script Bash* [1].

Processo de Criação de Ciclos Autônômicos

Um desenvolvedor, ao querer utilizar o *framework* proposto para a criação de um ciclo autônômico, deve estender os três principais componentes: monitor, analisador/planejador e o plano de ação, que é o código a ser processado pelo executor. Além disso o sensor deve ser criado como um cliente *socket* coletando dados e os enviando ao monitor (passo 1). Como também o atuador, que poderá ser qualquer aplicação responsável pela reconfiguração do elemento gerenciado (passo 2).

O monitor deve ser implementado (passo 3) de maneira a filtrar o que é relevante para a tomada de decisão. Ele receberá os dados vindos de uma conexão *socket* e seu código possuirá os filtros para esse conteúdo. Assim o monitor deve ser criado de forma a conhecer todos os possíveis dados que terá como entrada. Esse componente também deverá ser desenvolvido com conhecimento dos seus resultados,

para que não seja enviado dados irrelevantes ao próximo componente do ciclo (`Analyser`).

Em seguida deve ser implementado o componente de análise e planejamento (passo 4), que possuirá a estratégia autônoma. O algoritmo dessa última pode utilizar-se de técnicas de inteligência para a tomada de decisão, então esse componente é estendido de maneira livre, bastando que conheça os dados vindos do monitor e tenha como saída parâmetros para a criação do plano de ações, a ser passado para processamento pelo executor. No entanto, é possível que não tenha nada como saída, e nesse caso não será construído nenhum plano de ações.

A próxima etapa é montar o código do plano de ações (passo 5) nos quais o `Analyser` (componente de análise e planejamento) irá passar os parâmetros que equivalem a tomada de decisão. De acordo com esses parâmetros recebidos, será realizada uma ação diferente ou com configurações diferentes. Dessa maneira, todas as possíveis ações que poderão compor um plano de ações devem ser implementadas, de forma que a ação a ser executada seja escolhida em tempo de execução e os parâmetros passados para ela sejam dinâmicos, ambas decisões a escolha da estratégia autônoma. Essas ações devem ser também as responsáveis pela comunicação existente com o atuador, pois seus códigos serão processados pelo executor para a realização de alguma reconfiguração no elemento gerenciado.

Após as etapas descritas anteriormente, o que deve ser feito é o registro dos componentes na classe `RegisterManager` (passo 6). Como já explicado anteriormente, essa classe inicializa o *framework*, juntamente com todos os processos que a compõem. A funcionalidade dela é a criação do registro entre componentes, onde monitores são registrados no servidor, analisadores/planejadores nos monitores, que por sua vez são associados ao executor, de maneira que cada componente saiba para onde enviar o resultado de seu processamento, criando assim a ideia de ciclo.

6.2 Avaliação dos Ciclos Autônomicos

Nesta seção, descrevemos primeiramente como foram realizados os testes nas duas extensões e, em seguida, detalhamos os resultados obtidos. Os testes mostram como representamos a ideia dos ciclos autônomicos baseados em decepção

(Seções 5.2.2 e 5.2.3) e a descrição da implementação de cada componente do *framework*, para alcançar os resultados, os quais significam a obtenção de auto-segurança para a rede.

6.2.1 Testes

Primeiro Ciclo Autônômico

Nesse ciclo, o monitoramento é baseado em tempo, visto que o componente sensor faz uma verificação por novos registros no log do *honeypot* a cada período de tempo estabelecido por parâmetro. Em nossa implementação, estabelecemos a periodicidade em cinco segundos. Os dados sensoreados são logs de *honeypots* de baixa interatividade emulados com a ferramenta *Honeyd*, nos quais contém interações a nível de rede com qualquer usuário que acesse ou requisite serviços. A lógica do algoritmo do `Sensor` pode ser vista na Figura 6.3. Os dados contendo o log e o identificador do monitor são serializados para serem enviados.

```
1 enquanto true faça
2   | se newLogGenerated() então
3   |   | sendLog();
4   | senão
5   |   | sleep(5sec);
6   | fim
7 fim
```

Figura 6.3: Lógica do sensor no primeiro ciclo autônômico.

O componente `Monitor`, ao receber os logs, executa três métodos sobre o conteúdo. O primeiro é responsável por descartar todos os campos desnecessários para a estratégia autônômica. O segundo, exclui as linhas que contém os protocolos não utilizados também pela estratégia, dentre eles o IGMP, deixando apenas as que contém TCP, UDP, e ICMP. O último método envia as informações finais para a próxima fase do ciclo como um evento (`sendMonitoringEvent(filteredLog)`), de acordo

com os analisadores/planejadores registrados para recebê-los, após o processamento (filtragem) realizado pelos métodos anteriores.

Ao receber o evento contendo informações a serem processadas, o componente de análise e planejamento executa a estratégia autônômica baseada em intenções. Essa, por sua vez, é baseada em *ECA Rules* e a lógica da sua implementação pode ser vista na Figura 6.4.

```
1 para cada event faça
2   interactions[] = separatesInteractions(event);
3   para cada interaction faça
4     se !checkWhiteList(interaction, fileWhiteList) então
5       se checkServices(interaction, fileServices) então
6         services = examineServices(interaction, fileServices);
7         rules = generatesRules(interaction, services);
8         insertActionPlan(rules);
9       fim
10    fim
11  fim
12  se actionPlan != null então
13    sendPlanEvent(actionPlan);
14  fim
15 fim
```

Figura 6.4: Lógica da estratégia autônômica do primeiro ciclo.

Para cada evento recebido do monitor, as interações são separadas e armazenadas em um vetor, a fim de que possam ser analisadas individualmente. Para cada interação, é executado um método (*checkWhiteList(interaction, fileWhiteList)*) responsável por verificar se o campo do endereço IP, em sua versão quatro - *Internet Protocol version 4 (IPv4)*, de origem está contido na lista branca, sendo esta armazenada em um arquivo e que tem sua localização passada como parâmetro. O conteúdo desse arquivo é composto por uma lista de endereços considerados usuários legítimos da rede e, por isso, a estratégia autônômica

nunca gera regras que possam bloqueá-los. Após essa verificação, caso o IP não esteja na lista branca, é executado o método que checa os serviços contidos na rede (`checkServices(interaction, fileServices)`), retornando verdadeiro se houver pelo menos um serviço que funcione na mesma porta do *honeypot* que sofreu a interação. Dois parâmetros são passados para esse método: a própria interação e a localização do arquivo que contém informações dos serviços. Essas informações são, na verdade, configurações dos serviços que o administrador deseja proteger, e segue um formato contendo os seguintes campos:

- **IP:** Endereçamento IP do servidor;
- **Protocolo:** Protocolo utilizado pelo serviço, tendo como opções: TCP, UDP ou ICMP;
- **Porta:** Porta utilizada pelo serviço, sendo uma maneira de identificá-lo;
- **Comentários:** Usado para o administrador do sistema reconhecer mais facilmente os dispositivos.

Em uma configuração que especifique a utilização do protocolo ICMP, deverá ser informado o campo porta, porém ela será descartada no momento da criação da regra de *firewall*. O protocolo ICMP pertence a camada de rede do modelo TCP/IP [79] e não se utiliza de portas [8], por isso o descarte deste campo pela estratégia autônômica. Portanto, deverá ser informada alguma porta neste campo apenas para fins de formatação do arquivo de configuração.

Depois de passar por essas condições, é executado um método `examineServices(interaction, fileServices)` que faz algo semelhante ao anterior, no entanto, ele retorna para *services* o(s) endereço(s) de todos os serviços de produção que estejam no arquivo de configuração *fileServices* nos quais funcionem com mesma porta e protocolo. Em seguida *services* e a própria interação são passados para a criação e/ou exclusão das regras, de acordo com as listas cinza e negra, explicadas na Seção 5.2.2. Elas então são inseridas em um plano de ações, para que dessa forma, várias possam ser enviadas ao executor.

Por fim, é verificado se a tomada de decisão gerou alguma regra, para que o analisador/planejador não envie um evento vazio ou nulo (*null*) para o executor. Na

verdade, os eventos são as próprias regras, utilizadas como parâmetros na chamada do *Shell Script*. Esse script é escrito em *Bash* e, para este ciclo, ele é o atuador, responsável por reconfigurar um *firewall iptables*. Esse atuador também cria um relatório com horário e data de regras aplicadas, para que o administrador do sistema possa visualizar o que está sendo feito. A comunicação entre o *AutonomicSec* e atuador neste ciclo é via SSH.

Segundo Ciclo Autônômico

Este ciclo também serializa os dados no sensor para serem enviados, tem monitoramento baseado em tempo e utiliza *ECA Rules* para tomada de decisão, como no anterior. Os dados sensoreados são logs gerados pelo *iptables*, através de regras inseridas em sua configuração, vistas na Figura 6.5.

```
1 ...
2 iptables -A FORWARD -p tcp -i eth1 -m state --state ESTABLISHED,RELATED -j
  ACCEPT
3 iptables -A FORWARD -i eth1 -m state --state NEW -m limit --limit 10/hour
  --limit-burst 10 -j ACCEPT
4 iptables -A FORWARD -i eth1 -m state --state NEW -m limit --limit 1/hour --limit-burst
  1 -j LOG --log-prefix "Honeypot Comprometido " --log-level 7
5 ...
6 iptables -A FORWARD -i eth1 -m state --state NEW -j ACCEPT
```

Figura 6.5: Regras para controle de tráfego e log no *iptables*.

As linhas 3 e 4 estabelecem o registro no log, quando um dispositivo da rede interna inicia uma conexão com algum outro externo. No exemplo da Figura 6.5, a cada dez novas conexões por hora (linha 3) é criada uma ocorrência no log por hora (linha 4). Essa ocorrência é considerada um alerta pelo *iptables* e *syslog*, devido está sendo logada em prioridade de nível sete, com o parâmetro `--log-level 7`. Para identificar uma ocorrência no log criada por uma dessas regras, basta realizar uma busca pela palavra-chave Honeypot Comprometido. Uma regra específica para o protocolo TCP é necessária (linha 2), para que não sejam criadas ocorrências de conexões iniciadas da rede externa, através do seu estado ESTABLISHED,RELATED. O estado *related* ocorre quando uma conexão é nova, mas de alguma maneira é relacionado com outra já

estabelecida anteriormente e o *established* é para pacotes de uma mesma conexão já estabelecida. Essas principais regras foram baseadas nas utilizadas pelo *Honeywall* e *Sessionlimit*.

Quando as linhas chegam no monitor, ele extrai o endereço IP que gerou a ocorrência e o envia para o componente de análise e planejamento, como melhor explicado anteriormente na Seção 5.2.3. Nessa próxima fase é verificado se o endereço IP pertence a algum *honeypot* em um arquivo de configuração que contém informações da *honeynet* virtual. Caso seja, no mesmo arquivo de configuração são extraídos os demais campos para serem enviados como evento ao executor. A lógica do algoritmo da estratégia autônômica aplicada neste ciclo pode ser vista na Figura 6.6.

```
1 para cada event faça
2   | occurrences[] = separatesoccurrences(event);
3   | para cada occurrence faça
4   |   | se checkHoneypots(ip, fileHoneynet) então
5   |   |   | fields = extractFields(ip, fileHoneynet);
6   |   |   | insertActionPlan(fields);
7   |   |   | fim
8   |   | fim
9   |   | se actionPlan! = null então
10  |   |   | sendPlanEvent(actionPlan);
11  |   |   | fim
12  |   | fim
```

Figura 6.6: Lógica da estratégia autônômica do segundo ciclo.

Os campos contidos no arquivo de configuração (*fileHoneynet*) são listados abaixo:

- **IP do Hipervisor:** Endereço da máquina do hipervisor responsável pelo gerenciamento da virtualização;
- **IP do *honeypot*:** Endereço do *honeypot*, para sua identificação quando gerarem ocorrências;

- **Nome:** Possui o nome da máquina virtual no qual está funcionando o *honeypot*;
- **Número de sequência:** Número de sequência que conta a quantidade de vezes que um determinado *honeypot* virtual é substituído. Também utilizado pelo algoritmo do atuador para a criação de novo nome para a máquina virtual, concatenando-o com o nome anterior, por exemplo: Ubuntu3, Ubuntu4, Ubuntu5;
- **Localização da máquina:** Localização das imagens da máquina virtual que representa a réplica não infectada por códigos maliciosos.

Para os nossos testes utilizamos *honeypots* de alta interatividade, sendo instalado neles sistemas operacionais reais. Os *honeypots* são considerados de alta interatividade por serem máquinas virtuais que oferecem serviços e recursos de sistemas reais, e não emulados. Usamos como hipervisor o *VirtualBox* [11] e a ferramenta *VBoxManage* [10] para manipular as máquinas virtuais através de *Shell Script Bash*, sendo esse último a representação dos atuadores no ciclo. A comunicação entre o *AutonomicSec* e atuador neste ciclo é via SSH.

6.2.2 Resultados

Os resultados que apresentamos são organizados de maneira a mostrar a reconfiguração aplicada na rede, em seus sistemas de segurança, e o que foi produzido por cada fase do ciclo. Mostramos também os campos utilizados nos componentes que usam arquivos de configuração. O cenário utilizado para a coleta dos resultados pode ser visto na Figura 6.7.

As configurações principais de endereçamento desse cenário são:

- **Firewall**
 - **Interface da rede externa (eth0):** 192.168.177.2
 - **Interface da rede interna - DMZ (eth1):** 192.168.0.1
 - **Interface da rede interna (eth2):** 192.168.1.1
- **AutonomicSec:** 192.168.0.10

- **Honeynet Virtual**

- **Hipervisor:** 192.168.0.50
- **Máquina Virtual 1 - Honeypot de alta interatividade:** 192.168.0.201
- **Máquina Virtual 2 - Honeypot de alta interatividade:** 192.168.0.202
- **Honeypot de baixa interatividade 1:** 192.168.0.211
- **Honeypot de baixa interatividade 2:** 192.168.0.212
- **Honeypot de baixa interatividade 3:** 192.168.0.213

- **Servidores de Produção**

- **Servidor Web:** 192.168.0.120
- **Servidor DNS:** 192.168.0.121
- **Servidor SSH:** 192.168.0.130

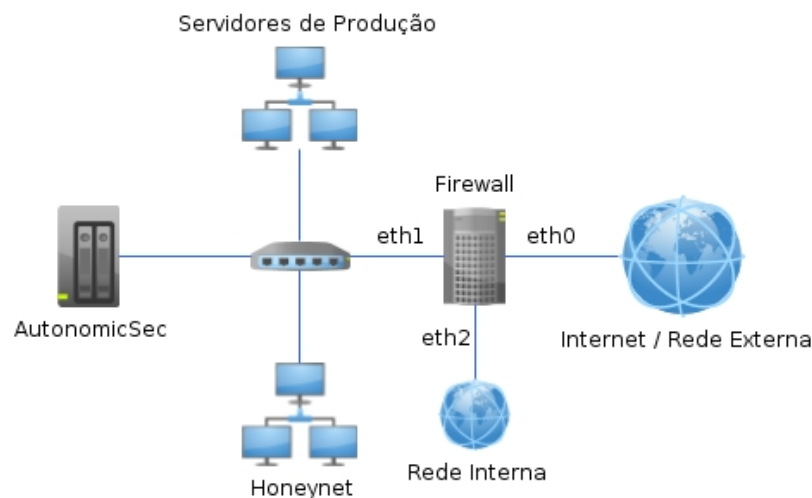


Figura 6.7: Cenário para coleta de resultados.

Primeiro Ciclo Autônomico

Inicialmente, em um sensor deste primeiro ciclo autônomico, foram gerados vários registros no log, nos quais selecionamos alguns para serem explorados e que podem ser vistos na Figura 6.8. O log segue o formato da ferramenta *Honeyd* e geraram três eventos pelo sensor para serem enviados ao monitor.

Nas linhas 1 e 2 foram registradas conexões *multicast*, sendo que na primeira foi de *Multicast Domain Name System* (mDNS) [4] e a segunda com o uso do protocolo IGMP. Nas linhas 3 e 4 são também conexões mDNS. As linha 5 a 8 são conexões ICMP e provavelmente de um ping iniciada por uma máquina com sistema operacional *Windows*, pois são quatro pacotes, em direção ao *honeypot* de baixa interatividade com endereço 192.168.0.212. As linhas 9 e 10 são requisições *Domain Name System* (DNS) para o *honeypot* de endereço 192.168.0.211. E nas linhas 11/12 e 13/14 são tentativas de conexão a servidores SSH e Web, respectivamente.

```
1 2012-01-09-13:50:04.2573 udp(17) - 192.168.0.50 5353 224.0.0.251 5353: 67
2 2012-01-09-13:51:32.7188 igmp(2) - 224.0.0.22 192.168.0.10: 40
3 2012-01-09-13:51:32.7588 udp(17) - 192.168.0.10 5353 224.0.0.251 5353: 210
4 2012-01-09-13:51:32.7704 udp(17) - 192.168.0.215 5353 224.0.0.251 5353: 305
5 2012-01-09-13:51:33.5703 icmp(1) - 192.168.177.247 192.168.0.212: 8(0): 84
6 2012-01-09-13:51:34.5704 icmp(1) - 192.168.177.247 192.168.0.212: 8(0): 84
7 2012-01-09-13:51:34.5785 icmp(1) - 192.168.177.247 192.168.0.212: 8(0): 84
8 2012-01-09-13:51:34.5825 icmp(1) - 192.168.177.247 192.168.0.212: 8(0): 84
9 2012-01-09-13:51:34.9720 udp(17) - 192.168.0.1 53 192.168.0.211 53: 71
10 2012-01-09-13:51:36.4733 udp(17) - 192.168.1.45 53 192.168.0.211 53: 71
11 2012-01-09-14:05:38.1172 tcp(6) S 192.168.177.247 1331 192.168.0.211 22
12 2012-01-09-14:05:39.1532 tcp(6) E 192.168.177.247 1331 192.168.0.211 22: 0 13
13 2012-01-09-14:13:27.1009 tcp(6) S 192.168.177.247 1550 192.168.0.213 80:
14 2012-01-09-14:13:28.1502 tcp(6) E 192.168.177.247 1550 192.168.0.213 80: 0
15 ...
```

Figura 6.8: Log gerado no *Honeypot*.

As linhas 1 a 10 representam o primeiro evento, 11 e 12 o segundo e, por fim, 13 e 14 o terceiro. Pois, como o sensoriamento/monitoramento é baseado em tempo e ocorre a cada cinco segundos neste ciclo, essas foram ocorrências em tempos distintos.

Ao chegar no componente de monitoramento, o log passa por um filtro que organiza as informações e descarta dados desnecessários. Dessa forma, os registros

mostrados anteriormente tem o resultado mostrado na Figura 6.9, após filtragem realizada pelo monitor.

```
1 udp 192.168.0.50 5353 224.0.0.251 5353
2 udp 192.168.0.10 5353 224.0.0.251 5353
3 udp 192.168.0.215 5353 224.0.0.251 5353
4 icmp 192.168.177.247 192.168.0.212
5 icmp 192.168.177.247 192.168.0.212
6 icmp 192.168.177.247 192.168.0.212
7 icmp 192.168.177.247 192.168.0.212
8 udp 192.168.0.1 53 192.168.0.211 53
9 udp 192.168.1.45 53 192.168.0.211 53
10 tcp 192.168.177.247 1331 192.168.0.211 22
11 tcp 192.168.177.247 1331 192.168.0.211 22
12 tcp 192.168.177.247 1550 192.168.0.213 80
13 tcp 192.168.177.247 1550 192.168.0.213 80
```

Figura 6.9: Informações resultantes do filtro realizado pelo monitor.

Neste resultado é possível observar que a linha que continha uma conexão com protocolo IGMP foi descartada, assim como campos ou caracteres não úteis para a estratégia autônômica. Ao chegar na fase de análise e planejamento, as informações serão usadas para uma tomada de decisão, na qual utiliza os arquivos *fileWhiteList* e *fileServices*, vistos nas Figuras 6.10 e 6.11, respectivamente.

```
1 192.168.0.1
2 192.168.0.10
3 192.168.0.50
4 192.168.0.201
5 192.168.0.202
6 192.168.0.211
7 192.168.0.212
8 192.168.0.213
9 192.168.0.120
10 192.168.0.121
11 192.168.0.130
```

Figura 6.10: Arquivo de configuração da lista branca.

Na lista branca adicionamos os endereços dos *hosts* da rede que consideramos confiáveis por não representarem uma ameaça. Então inicialmente colocamos os endereços do *firewall*, *AutonomicSec* e da máquina no qual tem o hipervisor das máquinas virtuais (linhas 1 a 3). Em seguida, os endereços das máquinas virtuais que estão os *honeypots* de baixa interatividade (4 e 5) e também os endereços dos próprios *honeypots* (6 a 8) e servidores de produção (9 a 11).

```
1 192.168.0.120 tcp 80 web
2 192.168.0.120 icmp 0 icmp-web
3 192.168.0.121 udp 53 dns
4 192.168.0.121 icmp 0 icmp-dns
5 192.168.0.130 tcp 22 ssh
```

Figura 6.11: Arquivo de configuração dos serviços.

No arquivo de configuração, que contém as informações dos servidores de produção com seus respectivos serviços, inserimos os dados dos três que compõem nosso cenário: Web (linhas 1 e 2), DNS (3 e 4) e SSH (5). Nas linhas 2 e 4 repetimos o ICMP para os dois servidores, pois os dois respondem a requisições desse protocolo. Essa configuração dos servidores de produção do nosso cenário pode ser representada na Figura 6.12.

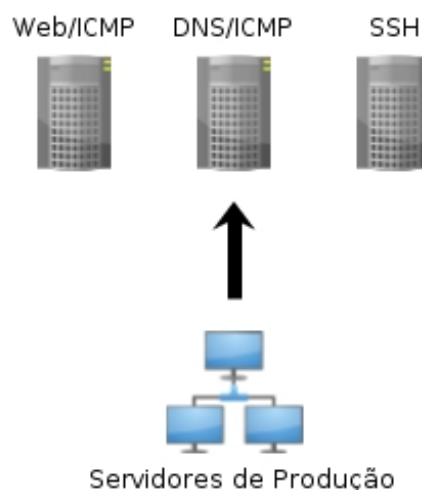


Figura 6.12: Servidores de produção do cenário.

Então é realizado o processamento da estratégia, na qual tivemos como resultado as regras do *firewall iptables* vistas na Figura 6.13. As linhas 1 a 3 são regras

geradas pelo primeiro evento, a 4 pelo segundo e a 5 pelo terceiro. A seguir os objetivos das regras e os motivos nos quais as geraram são explicados:

- **Regra 1:** Pacotes encaminhados pelo *firewall* com origem 192.168.177.247 e destino 192.168.0.120/24 com protocolo ICMP são rejeitados. Essa regra foi gerada porque houve interação do IP 192.168.177.247 com protocolo ICMP de algum *honeypot*;
- **Regra 2:** Pacotes encaminhados pelo *firewall* com origem 192.168.177.247 e destino 192.168.0.121/24 com protocolo ICMP são rejeitados. Essa regra foi gerada porque houve interação do IP 192.168.177.247 com protocolo ICMP de algum *honeypot*;
- **Regra 3:** Pacotes encaminhados pelo *firewall* com origem 192.168.1.45 e destino 192.168.0.121/24 com protocolo UDP e porta de destino 53 são rejeitados. Essa regra foi gerada porque houve interação do IP 192.168.1.45 com serviço de porta 53 que utiliza protocolo UDP de algum *honeypot*;
- **Regra 4:** Pacotes encaminhados pelo *firewall* com origem 192.168.177.247 e destino 192.168.0.130/24 com protocolo TCP e porta de destino 22 são rejeitados. Essa regra foi gerada porque houve interação do IP 192.168.177.247 com serviço de porta 22 que utiliza protocolo TCP de algum *honeypot*;
- **Regra 5:** Pacotes encaminhados pelo *firewall* com origem 192.168.177.247 e destino 192.168.0.120/24 com protocolo TCP e porta de destino 80 são rejeitados. Essa regra foi gerada porque houve interação do IP 192.168.177.247 com serviço de porta 80 que utiliza protocolo TCP de algum *honeypot*;

```
1 iptables -A FORWARD -s 192.168.177.247 -d 192.168.0.120/24 -p icmp -j REJECT
2 iptables -A FORWARD -s 192.168.177.247 -d 192.168.0.121/24 -p icmp -j REJECT
3 iptables -A FORWARD -s 192.168.1.45 -d 192.168.0.121/24 -p udp -dport 53 -j
  REJECT
4 iptables -A FORWARD -s 192.168.177.247 -d 192.168.0.130/24 -p tcp -dport 22 -j
  REJECT
5 iptables -A FORWARD -s 192.168.177.247 -d 192.168.0.120/24 -p tcp -dport 80 -j
  REJECT
```

Figura 6.13: Regras geradas pela estratégia baseada em intenções.

No entanto, quando a regra do terceiro evento é gerada (linha 5 da Figura 6.13), antes de ser realmente carregada para ser aplicada, um método verifica as ocorrências do endereço de origem em todas regras anteriormente criadas. Caso sejam encontradas mais de uma determinada quantidade, as ocorrências anteriores são excluídas e, logo após, é criada uma nova regra que bloqueia qualquer encaminhamento de pacote originado pelo endereço que excedeu as três ocorrências. Essa quantidade é um valor passado por parâmetro e que para nosso cenário utilizamos três (3), pois consideramos que mais de três ocorrências significa que o usuário quer realmente realizar atividade maliciosa na rede. Os endereços que são bloqueados dessa maneira, para toda rede, são os que compõem a lista negra. Com isso, após os três eventos, as regras otimizadas inseridas pelo atuador no *iptables* são as vistas na Figura 6.14.

```
1 iptables -A FORWARD -s 192.168.1.45 -d 192.168.0.121/24 -p udp --dport 53 -j  
  REJECT  
2 iptables -A FORWARD -s 192.168.177.247 -j REJECT
```

Figura 6.14: Regras otimizadas.

A otimização é feita através da exclusão de todas as regras geradas anteriormente que possuem o IP 192.168.177.247 (linhas 1, 2, 4 e 5 da Figura 6.13) e adição de uma nova que bloqueia o mesmo IP a toda rede (linha 2 da Figura 6.14). Nessa nova regra os pacotes encaminhados pelo *firewall* com origem 192.168.177.247 são rejeitados.

Segundo Ciclo Autônômico

Neste ciclo, como visto na Seção 6.2.1, são inseridas regras no *iptables* que criam logs. Com isso, foram gerados os registros no log vistos na Figura 6.15, os

quais foram identificados pela palavra-chave *Honeypot Comprometido* e, logo depois, passados para o monitor responsável por tratá-los.

```
1 ...
2 Jan 9 17:47:59 firewall-desktop kernel: [ 7459.494175] Honeypot Comprometido
  IN=eth1 OUT=eth0 SRC=192.168.0.201 DST=192.168.177.247 LEN=60
  TOS=0x00 PREC=0x00 TTL=63 ID=41750 DF PROTO=UDP SPT=60909 DPT=53
  LEN=40
3 Jan 9 18:51:27 firewall-desktop kernel: [11267.716532] Honeypot Comprometido
  IN=eth1 OUT=eth0 SRC=192.168.0.10 DST=200.137.130.2 LEN=58 TOS=0x00
  PREC=0x00 TTL=63 ID=32374 DF PROTO=UDP SPT=35161 DPT=53 LEN=38
4 Jan 9 20:33:19 firewall-desktop kernel: [17379.672687] Honeypot Comprometido
  IN=eth1 OUT=eth0 SRC=192.168.0.202 DST=192.168.177.247 LEN=84
  TOS=0x00 PREC=0x00 TTL=63 ID=0 DF PROTO=ICMP TYPE=8 CODE=0
  ID=1433 SEQ=1
5 ...
```

Figura 6.15: Log gerado no *Firewall*.

Ao chegar no monitor, o log é filtrado e extraído apenas o endereço de origem de cada registro. Então os endereços vistos na Figura 6.16 são passados para a etapa de análise e planejamento do ciclo.

```
1 192.168.0.201
2 192.168.0.10
3 192.168.0.202
```

Figura 6.16: Endereços após filtro realizado pelo monitor.

A estratégia deste ciclo é de verificar se o nó da rede que gerou o registro é um *honeypot*. Caso seja, campos do arquivo de configuração (Figura 6.17), usado na verificação para saber se é um *honeypot*, são enviados como parâmetro para o executor.

Esse último, por sua vez, irá passá-los como parâmetros ao atuador, para que ele possa realizar o processo de substituição da máquina virtual.

```
1 192.168.0.201 Ubuntu 0 /home/honeypot/VirtualBoxVMs/Ubuntu.vdi
2 192.168.0.202 Debian 2 /home/honeypot/VirtualBoxVMs/Debian.vdi
```

Figura 6.17: Arquivo de configuração dos *Honeypots*.

Como resultado de nosso teste, as ações geradas pela estratégia da fase de análise e planejamento são vistas na Figura 6.18. Na primeira linha, o atuador, quando receber o plano de ações, em sequência irá: desligar a máquina virtual com nome *Ubuntu*, excluir dados relacionados a ela, criar uma nova a partir da réplica ou imagem */home/honeypot/VirtualBoxVMs/Ubuntu.vdi* com nome *Ubuntu1* e inicializá-la. O mesmo processo é realizado na segunda linha, no entanto, será feita a substituição da máquina *Debian2* pela *Debian3*, com imagem localizada em */home/honeypot/VirtualBoxVMs/Debian.vdi*. Lembramos que esse atuador é escrito em *Shell Script Bash* e as tarefas realizadas por ele para a manipulação das máquinas virtuais utiliza a ferramenta *VBoxManage*.

```
1 Ubuntu 1 /home/honeypot/VirtualBoxVMs/Ubuntu.vdi
2 Debian 3 /home/honeypot/VirtualBoxVMs/Debian.vdi
```

Figura 6.18: Plano de ações criado pelo analisador/planejador.

6.3 Conclusão

Neste capítulo apresentamos a avaliação do *framework*, a qual detalhamos como foi feita a implementação do mesmo e como utilizá-lo para a criação de ciclos autônômicos. Logo depois, explicamos o desenvolvimento dos ciclos autônômicos, os testes realizados com eles e o resultados alcançados.

Como visto nos resultados, é possível obter integração e cooperação entre sistemas de segurança, que foram inicialmente desenvolvidos para funcionarem de forma isolada. Como também inteligência, através da implantação de estratégias

autônomicas que dinamizam o processo de proteção e autonomia para alcançar auto-segurança na rede.

7 Conclusões e Trabalhos Futuros

Em 2001 a IBM produziu um manifesto no qual alertou a dificuldade de gerenciamento dos sistemas computacionais atuais, apontando a autonomia como alternativa para a solução deste problema. Neste trabalho, apresentamos a definição e as principais características de uma nova abordagem para o desenvolvimento de sistemas que provem autonomia, a Computação Autônômica. Na qual envolve uma mudança no modo de projetar sistemas computacionais, pois esses devem possuir mecanismos efetivos que os permita monitorar, controlar e regular a si próprios, bem como recuperarem-se de problemas sem a necessidade de intervenções externas.

A ideia por trás dessa abordagem é desenvolver software auto-gerenciável, tendo pouca ou nenhuma intervenção humana para manter-se, baseado apenas em políticas de alto nível definidas pelo supervisor e no conhecimento adquirido ao longo do tempo. Uma série de propriedades autônômicas são utilizadas por esta abordagem para diminuir ou eliminar a intervenção humana sob a gerência dos sistemas computacionais, tais como: autocura, autoproteção, auto-otimização, autoaprendizagem, autoconfiguração, entre outras. Neste sentido, será colocado sob responsabilidade das próprias máquinas a tarefa de gerenciamento.

Apresentamos o estado atual da segurança de redes, no qual foi possível conhecer as fases que uma rede pode ser protegida, os atuais problemas de segurança enfrentados por ela e as possibilidades de uso de virtualização na área de segurança. Os trabalhos relacionados à nossa pesquisa mostram que a direção correta para as aplicações de segurança é a adoção de integração, cooperação e inteligência, para provê autonomia.

Dessa forma, mostramos que as redes de computadores, mais especificamente a gerência da segurança de redes, são cenários onde a CA pode ser aplicada, principalmente pelo crescimento resultante da Internet, que as tornam muito mais dinâmicas. Para isto foi explicitado a necessidade da segurança de redes por mecanismos autônômicos e maneiras de como é possível implementá-los. Isso

mostra que os recursos de autonomia fornecidos pela CA é o caminho mais viável para solucionar problemas de segurança existentes nas redes de computadores.

Contribuições

Nesse contexto, as principais contribuições desse trabalho são:

- A proposta de um mecanismo autônomo para segurança de redes baseado em decepção, na qual combina a ideia de CA com a utilização de *honeypots*. Esse mecanismo é composto de um *framework* baseado no modelo arquitetural MAPE-K e dois ciclos autônicos, que são extensões do *framework*. Através desse mecanismo integramos sistemas de segurança de maneira que eles cooperem entre si;
- O *framework* autônomo, seu modelo e implementação, para ser reutilizado em extensões que desejam criar outros ciclos autônicos;
- A avaliação do mecanismo autônomo proposto, através da realização de testes com os dois ciclos autônicos em cenário comum de uso. Nesta avaliação, detalhamos os resultados obtidos em cada componente correspondente a uma fase do ciclo MAPE-K. Ainda mostramos, através desses resultados, o que o mecanismo autônomo agrega para a segurança de redes;
- Uma discussão sobre a aplicabilidade de CA autônoma à área de segurança de redes, de maneira que foi possível contextualizar a nossa proposta, que combina CA com metodologias de decepção;
- Uma análise comparativa entre o mecanismo autônomo proposto e outros trabalhos relevantes, buscando caracterizá-lo em relação a esses últimos;
- Para contextualizar essas principais contribuições, ainda exploramos os tópicos de pesquisa fazendo:
 - Uma apresentação sobre Computação Autônoma, na qual mostramos sua origem e conceitos, principais propriedades, arquitetura, modelo MAPE-K e suas fases;
 - Uma visão geral do estado atual da segurança das redes de computadores.

Lições Aprendidas

No decorrer da pesquisa e desenvolvimento do trabalho aprendemos algumas lições, as quais listamos a seguir:

- A teoria de CA ainda está em desenvolvimento e seu objetivo principal é fornecer auto-gerenciamento para os sistemas computacionais, então a maneira que seus conceitos são interpretados e aplicados são particulares a cada situação. Desenvolvedores e pesquisadores podem ampliar os recursos que a CA fornece de acordo com a sua necessidade, buscando obter o melhor dessa abordagem para desenvolvimento de sistemas tem a oferecer;
- A ideia de fornecer recursos para que os próprios sistemas, em conjunto, possam garantir segurança à rede é uma maneira eficiente de aproveitar todas as qualidades que eles podem oferecer. O emprego de autonomia para o processo de defesa é fazer com que os sistemas utilizem esses recursos uns dos outros, de forma integrada e coordenada, para contornar problemas de segurança;
- A partir do momento que "ensinamos" a um software que um determinado evento é considerado uma atividade maliciosa e que é possível se defender de uma tal maneira, e ainda o informamos as estratégias utilizadas para realizar a proteção, atribuímos a ele a capacidade de auto-segurança;
- A CA aplicada às redes de computadores ou a segurança destas não vem substituir o papel do administrador de redes ou do gerente de segurança ou, de maneira geral, do responsável por essas áreas no segmento de Tecnologia da Informação das organizações. Acreditamos que a CA vem fornecer, na verdade, um recurso a mais para esse profissional, para que ele possa garantir maior qualidade no serviço prestado;
- Em relação aos conhecimentos técnicos aprendidos, afirmamos que um estudo aprofundado sobre uma ferramenta desejável a ser utilizada como sensor ou atuador, tendo foco ou não em segurança, é o caminho para que ela possa ser integrada a ciclos autônômicos junto ao *AutonomicSec*.

Limitações

Em nossa proposta identificamos as seguintes limitações:

- O *AutonomicSec* foi projetado para funcionar de modo centralizado. Contudo, essa estrutura é vulnerável, por apresentar um único ponto de falhas. Ou seja, o próprio *AutonomicSec* pode se tornar uma vulnerabilidade, caso o atacante tenha conhecimento sobre ele;
- A comunicação entre o *AutonomicSec* e os atuadores é feita de forma remota usando *Shell Script Bash* através de esquemas de autenticação com chaves assimétricas, o que limita o uso dos ciclos à ferramenta *ssh* e a sistemas operacionais baseados em Linux;
- Da mesma forma, há limitações em relação as ferramentas ou softwares utilizados para desenvolver o mecanismo: *Shell Script Bash*, *iptables*, servidor *SSH* e *VirtualBox*. De maneira geral, criando uma certa restrição para seu uso apenas com os softwares citados ou necessidade de adequação com outros.

Trabalhos Futuros

A partir deste trabalho inicial, identificamos diversas possibilidades de trabalhos futuros que poderiam ser desenvolvidos, tais como:

- Neste trabalho utilizamos para a tomada de decisão para os dois ciclos, no processo de análise e planejamento, representações do conhecimento baseadas em *ECA Rules*. No entanto, poderiam ser aplicadas outras técnicas de inteligência, tais como Funções de Utilidades e Aprendizagem por Reforço;
- No mesmo sentido, poderiam ser criadas outras estratégias mais sofisticadas para os ciclos;
- Como explicado nas limitações, a nossa proposta foi projetada para funcionar de modo centralizado, então trabalhos futuros poderiam investigar a possibilidade de extensão do *AutonomicSec* em uma abordagem distribuída;

- O mecanismo autônomo não leva em consideração a tolerância a falhas de si próprio, portanto, propor autocura para o *AutonomicSec* é uma interessante contribuição;
- Outras formas de comunicação poderiam ser estudadas entre o *AutonomicSec* e sensores/atuadores, podendo ser inserido um mecanismo de descoberta dinâmica de sensores e atuadores na rede;
- A extensão do *framework* com o uso de outros sensores e atuadores é um ótimo trabalho a ser desenvolvido dentro dessa área de pesquisa, inserindo novos sistemas de segurança para cooperação e, com isso, obter um aumento no nível de segurança da rede;
- Uma ampliação dos testes nos dois ciclos autônomos, podendo serem realizados em redes de alta velocidade para mensurar desempenho, ou testes de viabilidade com o uso de *Internet Protocol version 6* (IPv6) e com outras ferramentas diferentes das usadas neste trabalho;
- O nossa proposta pode ser estendida ou adaptada para uso junto a outras tecnologias e áreas de pesquisa, tais como: Redes de Sensores Sem Fio, Computação nas Nuvens, Computação em Grades, Computação Móvel, Computação Ubíqua, Computação Pervasiva ou Redes em Malha.

Referências Bibliográficas

- [1] Bash – gnu project – free software foundation. Disponível em: <http://www.gnu.org/software/bash>. Acessado em: 24 jan. 2012.
- [2] Eclipse – the eclipse foundation open source community website. Disponível em: <http://www.eclipse.org>. Acessado em: 23 jan. 2012.
- [3] Estatísticas do cert.br – incidentes. Disponível em: <http://www.cert.br/stats/incidentes>. Acessado em: 03 jan. 2012.
- [4] Multicast dns. Disponível em: <http://www.multicastdns.org>. Acessado em: 24 jan. 2012.
- [5] netfilter/iptables project homepage – the netfilter.org project. Disponível em: <http://www.netfilter.org>. Acessado em: 24 jan. 2012.
- [6] Official guardian web page – guardian active response for snort. Disponível em: <http://www.chaotic.org/guardian>. Acessado em: 03 jan. 2012.
- [7] Oracle e java – tecnologias. Disponível em: <http://www.oracle.com/br/technologies/java/index.html>. Acessado em: 23 jan. 2012.
- [8] Rfc 792 – internet control message protocol. Disponível em: <http://tools.ietf.org/html/rfc792>. Acessado em: 24 jan. 2012.
- [9] Ssh communications security, ssh, secure shell, data-in-transit. Disponível em: <http://ssh.com>. Acessado em: 24 jan. 2012.
- [10] Vboxmanage. Disponível em: <http://www.virtualbox.org/manual/ch08.html>. Acessado em: 24 jan. 2012.
- [11] Vm virtualbox. Disponível em: <http://www.virtualbox.org>. Acessado em: 24 jan. 2012.

- [12] M. Abu Rajab, J. Zarfoss, F. Monrose, and A. Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, IMC '06*, pages 41–52, New York, NY, USA, 2006. ACM.
- [13] C. Act-Net Consortium. The active database management system manifesto: a rulebase of adbms features. *SIGMOD Rec.*, 25:40–49, September 1996.
- [14] N. Agoulmine, S. Balasubramaniam, D. Botvich, J. Strassner, E. Lehtihet, and W. Donnelly. Challenges for autonomic network management. In *1st IEEE International Workshop on Modelling Autonomic Communications Environments - MACE*, 2006.
- [15] N. Al-Gharabally, N. El-Sayed, S. Al-Mulla, and I. Ahmad. Wireless honeypots: survey and assessment. In *Proceedings of the 2009 conference on Information Science, Technology and Applications, ISTA '09*, pages 45–52, New York, NY, USA, 2009. ACM.
- [16] S. Almotairi, A. Clark, G. Mohay, and J. Zimmermann. A technique for detecting new attacks in low-interaction honeypot traffic. In *Proceedings of the 2009 Fourth International Conference on Internet Monitoring and Protection*, pages 7–13, Washington, DC, USA, 2009. IEEE Computer Society.
- [17] M. F. A. Assunção. *Honeypots e honeynets: aprenda a detectar e enganar invasores*. Visual Books, Florianópolis, 2009.
- [18] S. Atay and M. Masera. Challenges for the security analysis of next generation networks. In *Broadband Communications, Networks, and Systems, 2009. BROADNETS 2009. Sixth International Conference on*, pages 1–4, sept. 2009.
- [19] T. Braga, F. Silva, L. Ruiz, and H. Assunção. Redes autonômicas. In *SBRC '06: Anais dos Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 159–208. SBC, 2006.
- [20] A. Castiglione, A. De Santis, U. Fiore, and F. Palmieri. An enhanced firewall scheme for dynamic and adaptive containment of emerging security threats. In *Broadband, Wireless Computing, Communication and Applications (BWCCA), 2010 International Conference on*, pages 475–481, nov. 2010.

- [21] G. Chamales. The honeywall cd-rom. *Security Privacy, IEEE*, 2(2):77–79, mar-apr 2004.
- [22] H. Chen, Y. Al-Nashif, G. Qu, and S. Hariri. Self-configuration of network security. In *Enterprise Distributed Object Computing Conference, 2007. EDOC 2007. 11th IEEE International*, page 97, oct. 2007.
- [23] S. Corrêa and R. Cerqueira. Computação autônoma: Conceitos, infra-estruturas e soluções em sistemas distribuídos. In *SBRC '09: Anais dos Minicursos do Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, pages 151–198. SBC, 2009.
- [24] R. P. da Cunha Neto. Sistema de detecção de intrusos em ataques oriundos de botnets utilizando método de detecção híbrido. Master's thesis, PPGEE/UFMA, 2011.
- [25] R. L. da Rocha Ataíde. Uma arquitetura para a detecção de intrusos no ambiente wireless usando redes neurais artificiais. Master's thesis, PPGEE/UFMA, 2007.
- [26] A. L. da Silva. Modelo ids para usuários de dispositivos móveis. Master's thesis, PPGEE/UFMA, 2008.
- [27] Y.-S. Dai, M. Hinchey, M. Qi, and X. Zou. Autonomic security and self-protection based on feature-recognition with virtual neurons. In *Proceedings of the 2nd IEEE International Symposium on Dependable, Autonomic and Secure Computing, DASC '06*, pages 227–234, Washington, DC, USA, 2006. IEEE Computer Society.
- [28] L. P. de Melo, D. M. Amaral, F. Sakakibara, A. R. de Almeida, R. T. de Sousa Junior, and A. C. A. Nascimento. Análise de malware: Investigação de códigos maliciosos através de uma abordagem prática. In *SBSeg '11: Anais dos Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 9–52. SBC, 2011.
- [29] X. Dong, S. Hariri, L. Xue, H. Chen, M. Zhang, S. Pavuluri, and S. Rao. Autonomia: an autonomic computing environment. In *Performance, Computing, and Communications Conference, 2003. Conference Proceedings of the 2003 IEEE International*, pages 61 – 68, april 2003.

- [30] M. Feily, A. Shahrestani, and S. Ramadass. A survey of botnet and botnet detection. In *Emerging Security Information, Systems and Technologies, 2009. SECURWARE '09. Third International Conference on*, pages 268–273, june 2009.
- [31] D. S. F. Filho, V. M. Afonso, V. F. Martins, A. R. A. Grégio, P. L. de Geus, M. Jino, and R. D. C. dos Santos. Técnicas para análise dinâmica de malware. In *SBSeg '11: Anais dos Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 104–144. SBC, 2011.
- [32] E. Freeman and E. Freeman. *Use a Cabeça! Padrões de Projetos – Design Patterns*. Alta Books, 2009.
- [33] P. García-Teodoro, J. Díaz-Verdejo, G. Maciá-Fernández, and E. Vázquez. Anomaly-based network intrusion detection: Techniques, systems and challenges. *Computers & Security*, 28(1–2):18–28, 2009.
- [34] Gleydson Mazili da Silva. *Guia Foca GNU/Linux*, 2010. Disponível em: <http://www.guiafoca.org/>. Acessado em: 31 mar. 2012.
- [35] J. F. Gonçalves. Introdução à computação autônoma e sua aplicação em ambientes computacionais distribuídos. Monografia do curso de Bacharelado em Ciência da Computação da Universidade Federal do Maranhão, São Luís, MA, 2010.
- [36] A. Goodloe and L. Pike. Monitoring distributed real-time systems: A survey and future directions. Technical Report NASA/CR-2010-216724, NASA Langley Research Center, 2010. Disponível em <https://www.cs.indiana.edu/~lepik/pub/survey.pdf>. Acessado em: 31 mar. 2012.
- [37] P. H. C. Guerra, D. Guedes, W. M. Jr., C. Hoepers, K. Steding-Jessen, and M. H. Chaves. Caracterização de encadeamento de conexões para envio de spams. In *27o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Recife, PE, 2009.
- [38] P. H. C. Guerra, M. T. Ribeiro, D. Guedes, W. M. Jr., C. Hoepers, K. Steding-Jessen, and M. H. Chaves. Identificação e caracterização de spammers a partir de listas de destinatários. In *28o Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos*, Gramado, RS, 2010.

- [39] S. Hallsteinsen. Madam - theory of adaptation. Technical report, SINTEF ICT, 2010.
- [40] S. Hariri, B. Khargharia, H. Chen, J. Yang, Y. Zhang, M. Parashar, and H. Liu. The autonomic computing paradigm. *Cluster Computing*, 9:5–17, January 2006.
- [41] S. Hariri, G. Qu, R. Modukuri, H. Chen, and M. Yousif. Quality-of-protection (qop)-an online monitoring and self-protection mechanism. *Selected Areas in Communications, IEEE Journal on*, 23(10):1983 – 1993, oct. 2005.
- [42] M. Henke, E. N. Clayton Santos, E. Feitosa, E. dos Santos, and E. Souto. Aprendizagem de máquina para segurança de computadores: Métodos e aplicações. In *SBSeg '11: Anais dos Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 53–103. SBC, 2011.
- [43] B. W. P. Hoelz, F. I. Mesquita, and P. Auler. Live forensics em ambiente microsoft windows. In *SBSeg '11: Anais dos Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*, pages 232–275. SBC, 2011.
- [44] C. Hoepers, K. Steding-Jessen, and A. Montes. Honeynets Applied to the CSIRT Scenario. In *Proceedings of the 15th Annual Computer Security Incident Handling Conference, to appear, Ottawa, Canada, June 2003*.
- [45] M. C. Huebscher and J. A. McCann. A survey of autonomic computing — degrees, models, and applications. *ACM Comput. Surv.*, 40:7:1–7:28, August 2008.
- [46] IBM. *An architectural blueprint for autonomic computing*, 2003.
- [47] P. Kabiri and A. A. Ghorbani. Research on intrusion detection and response: A survey. *International Journal of Network Security*, 1:84–102, 2005.
- [48] J. Kephart and D. Chess. The vision of autonomic computing. *Computer*, 36(1):41 – 50, jan 2003.
- [49] A. Khalid, M. Haye, M. Khan, and S. Shamail. Survey of frameworks, architectures and techniques in autonomic computing. In *Autonomic and Autonomous Systems, 2009. ICAS '09. Fifth International Conference on*, pages 220 –225, april 2009.

- [50] M. Krause and H. F. Tipton. *Handbook of Information Security Management*. Auerbach Publications, 1999.
- [51] C. Kreibich and J. Crowcroft. Honeycomb: creating intrusion detection signatures using honeypots. *SIGCOMM Comput. Commun. Rev.*, 34:51–56, January 2004.
- [52] W. Kun, W. Jin-dong, S. Liu-qing, and H. Zhen-hua. An intelligent security defensive software scheme and realization. In *Intelligent Information Technology and Security Informatics (IITSI), 2010 Third International Symposium on*, pages 793–796, april 2010.
- [53] M. A. P. Laureano and C. A. Maziero. Virtualização: Conceitos e aplicações em segurança. In *SBSeg '08: Anais dos Minicursos do Simpósio Brasileiro em Segurança da Informação e de Sistemas Computacionais*. SBC, 2008.
- [54] F. d. Á. Leão Moraes. Segurança e confiabilidade em ids baseados em agentes. Master's thesis, PPGEE/UFMA, 2009.
- [55] M. Léger, T. Ledoux, and T. Coupaye. Reliable dynamic reconfigurations in the fractal component model. In *Proceedings of the 6th international workshop on Adaptive and reflective middleware: held at the ACM/IFIP/USENIX International Middleware Conference, ARM '07*, pages 3:1–3:6, New York, NY, USA, 2007. ACM.
- [56] C. F. L. Lima. Agentes inteligentes para detecção de intrusos em redes de computadores. Master's thesis, PPGEE/UFMA, 2002.
- [57] A. Mairh, D. Barik, K. Verma, and D. Jena. Honeypot in network security: a survey. In *Proceedings of the 2011 International Conference on Communication, Computing & Security, ICCCS '11*, pages 600–605, New York, NY, USA, 2011. ACM.
- [58] M. Mansouri-Samani. *Monitoring of Distributed Systems*. PhD thesis, University of London. Imperial College of Science, Technology and Medicine, December 1995.
- [59] A. Marcelo and M. Pitanga. *Honeypots: a arte de iludir hackers*. Brasport, Rio de Janeiro, 2003.
- [60] P. K. McKinley, S. M. Sadjadi, E. P. Kasten, and B. H. C. Cheng. Composing adaptive software. *Computer*, 37:56–64, July 2004.

- [61] S. Mirzaie, A. Elyato, and M. Sarram. Preventing of syn flood attack with iptables firewall. In *Communication Software and Networks, 2010. ICCSN '10. Second International Conference on*, pages 532–535, 2010.
- [62] R. Muraleedharan and L. Osadciw. An intrusion detection framework for sensor networks using honeypot and swarm intelligence. In *Mobile and Ubiquitous Systems: Networking Services, MobiQuitous, 2009. MobiQuitous '09. 6th Annual International*, pages 1–2, july 2009.
- [63] R. Murch. *Autonomic Computing*. IBM Press/Prentice-Hall, 1 edition, 2004.
- [64] T. J. O'Connor and B. Sangster. honeym: a framework for implementing virtual honeyclients for mobile devices. In *Proceedings of the third ACM conference on Wireless network security, WiSec '10*, pages 129–138, New York, NY, USA, 2010. ACM.
- [65] A. A. P. Oliveira. Sociedade de agentes para a monitoração de ataques e respostas automatizadas. Master's thesis, PPGEE/UFMA, 2005.
- [66] N. D. Palma, D. P. Laumay, and L. Bellissard. Ensuring dynamic reconfiguration consistency. In *In 6th International Workshop on Component-Oriented Programming (WCOP 2001), ECOOP related Workshop*, pages 18–24, 2001.
- [67] G. Palmer. A road map for digital forensic research. Technical report, First Digital Forensic Research Workshop (DFRWS), 2001.
- [68] M. Parashar and S. Hariri. Autonomic computing: An overview. In *Unconventional Programming Paradigms*, pages 247–259. Springer Verlag, 2005.
- [69] M. Parashar and S. Hariri. *Autonomic Computing: Concepts, Infrastructure, and Applications*. CRC Press, 2006.
- [70] V. Paxson. Bro: A system for detecting network intruders in real-time. In *Computer Networks*, pages 2435–2463, 1999.
- [71] E. S. Pilli, R. C. Joshi, and R. Niyogi. Network forensic frameworks: Survey and research challenges. *Digital Investigation*, 7(1-2):14–27, 2010.

- [72] G. Portokalidis and H. Bos. Sweetbait: Zero-hour worm detection and containment using low- and high-interaction honeypots. *Comput. Netw.*, 51:1256–1274, April 2007.
- [73] G. Portokalidis, A. Slowinska, and H. Bos. Argos: an emulator for fingerprinting zero-day attacks for advertised honeypots with automatic signature generation. *SIGOPS Oper. Syst. Rev.*, 40:15–27, April 2006.
- [74] S. Poslad. *Autonomous Systems and Artificial Life*, pages 317–341. John Wiley & Sons, Ltd, 2009.
- [75] N. Provos. A virtual honeypot framework. In *Proceedings of the 13th conference on USENIX Security Symposium, SSYM'04*, pages 1–1, Berkeley, CA, USA, 2004. USENIX Association.
- [76] M. Qassrawi and Z. Hongli. Deception methodology in virtual honeypots. In *Networks Security Wireless Communications and Trusted Computing (NSWCTC), 2010 Second International Conference on*, volume 2, pages 462 –467, april 2010.
- [77] G. Qu, S. Hariri, S. Jangiti, J. Rudraraju, S. Oh, S. Fayssal, G. Zhang, and M. Parashar. Online monitoring and analysis for self-protection against network attacks. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 324 – 325, may 2004.
- [78] M. Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX conference on System administration, LISA '99*, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.
- [79] K. W. Ross and J. F. Kurose. *Redes de Computadores e a Internet: Uma abordagem top-down*. Addison Wesley, 3 edition, 2006.
- [80] A. Saxena, M. Lacoste, T. Jarboui, U. Lücking, and B. Steinke. A software framework for autonomic security in pervasive environments. In *Proceedings of the 3rd international conference on Information systems security, ICISS'07*, pages 91–109, Berlin, Heidelberg, 2007. Springer-Verlag.
- [81] L. Shen, J. Wang, K. Wang, and H. Zhang. The design of intelligent security defensive software based on autonomic computing. In *Proceedings of the 2009*

- Second International Conference on Intelligent Computation Technology and Automation - Volume 01*, pages 489–491, Washington, DC, USA, 2009. IEEE Computer Society.
- [82] J. E. Smith and R. Nair. *Virtual Machines: Architectures, Implementations and Applications*. Morgan Kaufmann, 2004.
- [83] J. E. Smith and R. Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, may 2005.
- [84] A. Soares Teles, F. J. da Silva e Silva, and Z. Abdelouahab. Computação autônômica aplicada a segurança de redes. In *V Escola Regional de Informática Ceará-Maranhão-Piauí*, pages 54–78. Sociedade Brasileira de Computação, 2011.
- [85] J. Song, H. Takakura, and Y. Kwon. A generalized feature extraction scheme to detect 0-day attacks via ids alerts. In *Applications and the Internet, 2008. SAINT 2008. International Symposium on*, pages 55 –61, 28 2008-aug. 1 2008.
- [86] L. Spitzner. Honeypots: Definitions and value of honeypots. In *SANS Annual Conference*, 2002.
- [87] W. Stallings. *Criptografia e segurança de redes: Princípios e práticas*. Prentice Hall, 4 edition, 2008.
- [88] K. Steding-Jessen and C. Hoepers. Sessionlimit. Disponível em: <http://www.honeynet.org.br/tools/sessionlimit/sessionlimit-0.3.README>. Acessado em: 03 jan. 2012.
- [89] A. S. Tanenbaum. *Sistemas Operacionais Modernos*. Prentice Hall Brasil, 2006.
- [90] X. Tang. The generation of attack signatures based on virtual honeypots. In *Parallel and Distributed Computing, Applications and Technologies (PDCAT), 2010 International Conference on*, pages 435 –439, dec. 2010.
- [91] A. S. Teles, J. P. M. Mendes, and Z. Abdelouahab. Autonomic computing applied to network security: A survey. *Cyber Journals: Multidisciplinary Journals in Science and Technology, Journal of Selected Areas in Telecommunications (JSAT)*, November Edition:7–14, 2011.
- [92] T. Thomas. *Segurança de Redes - Primeiros Passos*. Ciência Moderna - LCM, 2007.

- [93] V. Verendel, D. Nilsson, U. Larson, and E. Jonsson. An approach to using honeypots in in-vehicle networks. In *Vehicular Technology Conference, 2008. VTC 2008-Fall. IEEE 68th*, pages 1–5, sept. 2008.
- [94] B. Wang, P. Zhu, Q. Wen, and X. Yu. A honeynet-based firewall scheme with initiative security strategies. In *Computer Network and Multimedia Technology, 2009. CNMT 2009. International Symposium on*, pages 1–4, jan. 2009.
- [95] L. Xuanmin, S. Chang, Moses, and G. Jingyuan. Research on ip address replacement technology based on iptables. In *Wireless Communications, Networking and Mobile Computing (WiCOM), 2011 7th International Conference on*, pages 1–3, 2011.
- [96] K. Zolfaghar and S. Mohammadi. Securing bluetooth-based payment system using honeypot. In *Innovations in Information Technology, 2009. IIT '09. International Conference on*, pages 21–25, dec. 2009.
- [97] T. Zseby, H. Pfeffer, and S. Steglich. Concepts for self-protection. In Y. Zhang, L. T. Yang, and M. K. Denko, editors, *Autonomic Computing and Networking*, pages 355–380. Springer US, 2009.